Robert Meersman
Zahir Tari (Eds.)

LNCS 4803

On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS

OTM Confederated International Conferences
CoopIS, DOA, ODBASE, GADA, and IS 2007
Vilamoura, Portugal, November 2007
Proceedings, Part I

1 Part I

CoopIS DOA

GADA ODBASE

Springer

Robert Meersman Zahir Tari (Eds.)

On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS

OTM Confederated International Conferences
CoopIS, DOA, ODBASE, GADA, and IS 2007
Vilamoura, Portugal, November 25-30, 2007
Proceedings, Part I

Springer

Volume Editors

Robert Meersman
Vrije Universiteit Brussel (VUB), STARLab
Bldg G/10, Pleinlaan 2, 1050 Brussels, Belgium
E-mail: meersman@vub.ac.be

Zahir Tari
RMIT University, School of Computer Science and Information Technology
Bld 10.10, 376-392 Swanston Street, VIC 3001, Melbourne, Australia
E-mail: zahir.tari@rmit.edu.au

Library of Congress Control Number: 2007939491

CR Subject Classification (1998): H.2, H.3, H.4, C.2, H.5, D.2.12, I.2, K.4

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-76846-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-76846-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12192205 06/3180 5 4 3 2 1 0

Volume Editors

Robert Meersman
Zahir Tari

CoopIS

Francisco Curbera
Frank Leymann
Mathias Weske

DOA

Pascal Felber
Aad van Moorsel
Calton Pu

ODBASE

Tharam Dillon
Michele Missikoff
Steffen Staab

GADA

Pilar Herrero
Daniel S. Katz
María S. Pérez
Domenico Talia

IS

Mário Freire
Simão Melo de Sousa
Vitor Santos
Jong Hyuk Park

OTM 2007 General Co-chairs' Message

OnTheMove 2007 held in Vilamoura, Portugal, November 25–30 further consolidated the growth of the conference series that was started in Irvine, California in 2002, and then held in Catania, Sicily in 2003, in Cyprus in 2004 and 2005, and in Montpellier last year. It continues to attract a diversifying and representative selection of today's worldwide research on the scientific concepts underlying new computing paradigms that of necessity must be distributed, heterogeneous and autonomous yet meaningfully collaborative.

Indeed, as such large, complex and networked intelligent information systems become the focus and norm for computing, it is clear that there is an acute and increasing need to address and discuss in an integrated forum the implied software and system issues as well as methodological, semantical, theoretical and application issues. As we all know, e-mail, the Internet, and even video conferences are not sufficient for effective and efficient scientific exchange. This is why the OnTheMove (OTM) Federated Conferences series has been created to cover the increasingly wide yet closely connected range of fundamental technologies such as data and Web semantics, distributed objects, Web services, databases, information systems, workflow, cooperation, ubiquity, interoperability, mobility, grid and high-performance systems. OnTheMove aspires to be a primary scientific meeting place where all aspects of the development of Internet- and Intranet-based systems in organizations and for e-business are discussed in a scientifically motivated way. This sixth 2007 edition of the OTM Federated Conferences event, therefore, again provided, an opportunity for researchers and practitioners to understand and publish these developments within their individual as well as within their broader contexts.

Originally the federative structure of OTM was formed by the co-location of three related, complementary and successful main conference series: DOA (Distributed Objects and Applications, since 1999), covering the relevant infrastructure-enabling technologies, ODBASE (Ontologies, DataBases and Applications of SEmantics, since 2002) covering Web semantics, XML databases and ontologies, CoopIS (Cooperative Information Systems, since 1993) covering the application of these technologies in an enterprise context through, e.g., workflow systems and knowledge management. In 2006 a fourth conference, GADA (Grid computing, high-performAnce and Distributed Applications), was added as a main symposium, and this year the same happened with IS (Information Security). Both started off as successful workshops at OTM, the first covering the large-scale integration of heterogeneous computing systems and data resources with the aim of providing a global computing space, the second covering the issues of security in complex Internet-based information systems. Each of these five conferences encourages researchers to treat their respective topics within a framework that incorporates jointly (a) theory , (b) conceptual design and development, and (c) applications, in particular case studies and industrial solutions.

Following and expanding the model created in 2003, we again solicited and selected quality workshop proposals to complement the more "archival" nature of the main conferences with research results in a number of selected and more "avant garde" areas related to the general topic of distributed computing. For instance, the so-called Semantic Web has given rise to several novel research areas combining linguistics, information systems technology, and artificial intelligence, such as the modeling of (legal) regulatory systems and the ubiquitous nature of their usage. We were glad to see that no less than eight of our earlier successful workshops (notably AweSOMe, CAMS, SWWS, ORM, OnToContent, MONET, PerSys, RDDS) re-appeared in 2007 with a second or third edition, and that four brand-new workshops emerged to be selected and hosted, and were successfully organized by their respective proposers: NDKM, PIPE, PPN, and SSWS. We know that as before, workshop audiences will productively mingle with another and with those of the main conferences, as is already visible from the overlap in authors! The OTM organizers are especially grateful for the leadership and competence of Pilar Herrero in managing this complex process into a success for the fourth year in a row.

A special mention for 2007 is to be made of the third and enlarged edition of the OnTheMove Academy (formerly called Doctoral Consortium Workshop), our "vision for the future" in research in the areas covered by OTM. Its 2007 organizers, Antonia Albani, Torben Hansen and Johannes Maria Zaha, three young and active researchers, guaranteed once more the unique interactive formula to bring PhD students together: research proposals are submitted for evaluation; selected submissions and their approaches are presented by the students in front of a wider audience at the conference, and are independently and extensively analyzed and discussed in public by a panel of senior professors. This year these were once more Johann Eder and Maria Orlowska, under the guidance of Jan Dietz, the incumbent Dean of the OnTheMove Academy. The successful students only pay a minimal fee for the Doctoral Symposium itself and also are awarded free access to all other parts of the OTM program (in fact their attendance is largely sponsored by the other participants!).

All five main conferences and the associated workshops share the distributed aspects of modern computing systems, and the resulting application-pull created by the Internet and the so-called Semantic Web. For DOA 2007, the primary emphasis stayed on the distributed object infrastructure; for ODBASE 2007, it became the knowledge bases and methods required for enabling the use of formal semantics; for CoopIS 2007, the topic as usual was the interaction of such technologies and methods with management issues, such as occur in networked organizations; for GADA 2007, the topic was the scalable integration of heterogeneous computing systems and data resources with the aim of providing a global computing space; and last but not least in the relative newcomer IS 2007 the emphasis was on information security in the networked society. These subject areas overlap naturally and many submissions in fact also treated an envisaged mutual impact among them. As for the earlier editions, the organizers wanted to stimulate this cross-pollination by a shared program of famous keynote speakers: this year we were proud to announce Mark Little of Red Hat, York Sure of SAP Research, Donald Ferguson of Microsoft, and Dennis Gannon of Indiana University. As always, we also encouraged multiple event attendance by providing all authors, also those of workshop papers, with free access or discounts to one other conference or workshop of their choice.

We received a total of 362 submissions for the five main conferences and 241 for the workshops. Not only may we indeed again claim success in attracting an increasingly representative volume of scientific papers, but such a harvest of course allows the Program Committees to compose a higher-quality cross-section of current research in the areas covered by OTM. In fact, in spite of the larger number of submissions, the Program Chairs of each of the three main conferences decided to accept only approximately the same number of papers for presentation and publication as in 2004 and 2005 (i.e., average one paper out of every three to four submitted, not counting posters). For the workshops, the acceptance rate varied but was much stricter than before, consistently about one accepted paper for every two to three submitted. Also for this reason, we separate the proceedings into four books with their own titles, two for main conferences and two for workshops, and we are grateful to Springer for their suggestions and collaboration in producing these books and CD-Roms. The reviewing process by the respective Program Committees was again performed very professionally and each paper in the main conferences was reviewed by at least three referees, with arbitrated e-mail discussions in the case of strongly diverging evaluations. It may be worthwhile to emphasize that it is an explicit OnTheMove policy that all conference Program Committees and Chairs make their selections completely autonomously from the OTM organization itself. Continuing a costly but nice tradition, the OnTheMove Federated Event organizers decided again to make all proceedings available to all participants of conferences and workshops, independently of one's registration to a specific conference or workshop. Each participant also received a CD-Rom with the full combined proceedings (conferences + workshops).

The General Chairs are once more especially grateful to all the many people directly or indirectly involved in the set-up of these federated conferences, who contributed to making them a success. Few people realize what a large number of individuals have to be involved, and what a huge amount of work, and sometimes risk, the organization of an event like OTM entails. Apart from the persons mentioned above, we therefore in particular wish to thank our 12 main conference PC Co-chairs (GADA 2007: Pilar Herrero, Daniel Katz, María S. Pérez, Domenico Talia; DOA 2007: Pascal Felber, Aad van Moorsel, Calton Pu; ODBASE 2007: Tharam Dillon, Michele Missikoff, Steffen Staab; CoopIS 2007: Francisco Curbera, Frank Leymann, Mathias Weske; IS 2007: Mário Freire, Simão Melo de Sousa, Vitor Santos, Jong Hyuk Park) and our 36 workshop PC Co-chairs (Antonia Albani, Susana Alcalde, Adezzine Boukerche, George Buchanan, Roy Campbell, Werner Ceusters, Elizabeth Chang, Antonio

Coronato, Simon Courtenage, Ernesto Damiani, Skevos Evripidou, Pascal Felber, Fernando Ferri, Achille Fokoue, Mario Freire, Daniel Grosu, Michael Gurstein, Pilar Herrero, Terry Halpin, Annika Hinze, Jong Hyuk Park, Mustafa Jarrar, Jiankun Hu, Cornel Klein, David Lewis, Arek Kasprzyk, Thorsten Liebig, Gonzalo Méndez, Jelena Mitic, John Mylopoulos, Farid Nad-Abdessalam, Sjir Nijssen, the late Claude Ostyn, Bijan Parsia, Maurizio Rafanelli, Marta Sabou, Andreas Schmidt, Simão Melo de Sousa, York Sure, Katia Sycara, Thanassis Tiropanis, Arianna D'Ulizia, Rainer Unland, Eiko Yoneki, Yuanbo Guo).

All together with their many PC members, did a superb and professional job in selecting the best papers from the large harvest of submissions.

We also must heartily thank Jos Valente de Oliveira for the efforts in arranging facilities at the venue and coordinating the substantial and varied local activities needed for a multi-conference event such as ours. And we must all be grateful also to Ana Cecilia Martinez-Barbosa for researching and securing the sponsoring arrangements, to our extremely competent and experienced Conference Secretariat and technical support staff in Antwerp, Daniel Meersman, Ana-Cecilia, and Jan Demey, and last but not least to our energetic Publications Chair and loyal collaborator of many years in Melbourne, Kwong Yuen Lai, this year vigorously assisted by Vidura Gamini Abhaya and Peter Dimopoulos.

The General Chairs gratefully acknowledge the academic freedom, logistic support and facilities they enjoy from their respective institutions, Vrije Universiteit Brussel (VUB) and RMIT University, Melbourne, without which such an enterprise would not be feasible.

We do hope that the results of this federated scientific enterprise contribute to your research and your place in the scientific network.

August 2007

Robert Meersman
Zahir Tari

Organizing Committee

The OTM (On The Move) 2007 Federated Conferences, which involve CoopIS (Cooperative Information Systems), DOA (distributed Objects and Applications), GADA (Grid computing, high-performAnce and Distributed Applications), IS (Information Security) and ODBASE (Ontologies, Databases and Applications of Semantics) are proudly **supported** by RMIT University (School of Computer Science and Information Technology) and Vrije Universiteit Brussel (Department of Computer Science).

Executive Committee

OTM 2007 General Co-chairs	Robert Meersman (Vrije Universiteit Brussel, Belgium) and Zahir Tari (RMIT University, Australia)
GADA 2007 PC Co-chairs	Pilar Herrero (Universidad Politécnica de Madrid, Spain), Daniel Katz (Louisiana State University, USA), María S. Pérez (Universidad Politécnica de Madrid, Spain), and Domenico Talia (Università della Callabria, Italy)
CoopIS 2007 PC Co-chairs	Francisco Curbera (IBM, USA), Frank Leymann (University of Stuttgart, Germany), and Mathias Weske (University of Potsdam, Germany)
DOA 2007 PC Co-chairs	Pascal Felber (Université de Neuchâtel, Switzerland), Aad van Moorsel (Newcastle University, UK), and Calton Pu (Georgia Tech, USA)
IS 2007 PC Co-chairs	Mário M. Freire (University of Beira Interior, Portugal), Simão Melo de Sousa (University of Beira Interior, Portugal), Vitor Santos (Microsoft, Portugal), and Jong Hyuk Park (Kyungnam University, Korea)
ODBASE 2007 PC Co-chairs	Tharam Dillon (University of Technology Sydney, Australia), Michele Missikoff (CNR, Italy), and Steffen Staab (University of Koblenz-Landau, Germany)
Publication Co-chairs	Kwong Yuen Lai (RMIT University, Australia) and Vidura Gamini Abhaya (RMIT University, Australia)
Local Organizing Chair	José Valente de Oliveira (University of Algarve, Portugal)

Conferences Publicity Chair	Jean-Marc Petit (INSA, Lyon, France)
Workshops Publicity Chair	Gonzalo Mendez (Universidad Complutense de Madrid, Spain)
Secretariat	Ana-Cecilia Martinez Barbosa, Jan Demey, and Daniel Meersman

CoopIS 2007 Program Committee

Marco Aiello
Bernd Amann
Alistair Barros
Zohra Bellahsene
Boualem Benatallah
Salima Benbernou
Djamal Benslimane
Klemens Böhm
Laura Bright
Christoph Bussler
Malu Castellanos
Vincenco D'Andrea
Umesh Dayal
Susanna Donatelli
Marlon Dumas
Schahram Dustdar
ohannesson Eder
Rik Eshuis
Opher Etzion
Klaus Fischer
Avigdor Gal
Paul Grefen
Mohand-Said Hacid
Geert-Jan Houben
Michael Huhns
Paul Johannesson
Dimka Karastoyanova
Rania Khalaf
Bernd Krämer
Akhil Kumar
Dominik Kuropka
Tiziana Margaria
Maristella Matera
Massimo Mecella
Ingo Melzer
Jörg Müller
Wolfgang Nejdl
Werner Nutt
Andreas Oberweis
Mike Papazoglou
Cesare Pautasso
Barbara Pernici
Frank Puhlmann
Manfred Reichert
Stefanie Rinderle
Rainer Ruggaber
Kai-Uwe Sattler
Ralf Schenkel
Timos Sellis
Brigitte Trousse
Susan Urban
Willem-Jan Van den Heuvel
Wil Van der Aalst
Maria Esther Vidal
Jian Yang
Kyu-Young Whang
Leon Zhao
Michael zur Muehlen

DOA 2007 Program Committee

Marco Aiello
Bernd Amann
Alistair Barros,
Zohra Bellahsene
Boualem Benatallah
Salima Benbernou

Djamal Benslimane
Klemens Böhm
Laura Bright
Christoph Bussler
Malu Castellanos
Vincenco D'Andrea
Umesh Dayal
Susanna Donatelli
Marlon Dumas
Schahram Dustdar
Johannesson Eder
Rik Eshuis
Opher Etzion
Klaus Fischer
Avigdor Gal
Paul Grefen
Mohand-Said Hacid
Geert-Jan Houben
Michael Huhns
Paul Johannesson
Dimka Karastoyanova
Rania Khalaf
Bernd Krämer
Akhil Kumar
Dominik Kuropka
Tiziana Margaria
Maristella Matera
Massimo Mecella
Ingo Melzer
Jörg Müller
Wolfgang Nejdl
Werner Nutt
Andreas Oberweis
Mike Papazoglou
Cesare Pautasso
Barbara Pernici
Frank Puhlmann
Manfred Reichert
Stefanie Rinderle
Rainer Ruggaber
Kai-Uwe Sattler
Ralf Schenkel
Timos Sellis
Brigitte Trousse
Susan Urban
Willem-Jan Van den Heuvel,
Wil Van der Aalst
Maria Esther Vidal
Jian Yang
Kyu-Young Whang
Leon Zhao
Michael zur Muehlen

GADA 2007 Program Committee

Jemal Abawajy
Akshai Aggarwal
Sattar B. Sadkhan Almaliky
Artur Andrzejak
Amy Apon
Oscar Ardaiz
Costin Badica
Rosa M. Badia
Mark Baker
Angelos Bilas
Jose L. Bosque
Juan A. Bota Blaya
Pascal Bouvry
Rajkumar Buyya
Santi Caball Llobet
Mario Cannataro
Jesús Carretero
Charlie Catlett
Pablo Chacin
Isaac Chao
Jinjun Chen
Félix J. García Clemente
Carmela Comito
Toni Cortes
Geoff Coulson
Jose Cunha
Ewa Deelman
Marios Dikaiakos
Beniamino Di Martino
Jack Dongarra

Markus Endler
Alvaro A.A. Fernandes
Maria Ganzha
Felix García
Angel Lucas Gonzalez
Alastair Hampshire
Jose Cunha
Neil P Chue Hong
Eduardo Huedo
Jan Humble
Liviu Joita
Kostas Karasavvas
Chung-Ta King
Kamil Kuliberda
Laurent Lefevre
Ignacio M. Llorente
Francisco Luna
Edgar Magana
Gregorio Martinez
Ruben S. Montero
Reagan Moore
Mirela Notare
Hong Ong
Mohamed Ould-Khaoua
Marcin Paprzycki
Manish Parashar
Jose M. Peña
Dana Petcu
Beth A. Plale
José Luis Vázquez Poletti
María Eugenia de Pool
Bhanu Prasad
Thierry Priol
Víctor Robles
Rizos Sakellariou, Univ. of Manchester
Manuel Salvadores
Alberto Sanchez
Hamid Sarbazi-Azad
Franciszek Seredynski
Francisco José da Silva e Silva
Antonio F. Gómez Skarmeta
Enrique Soler
Heinz Stockinger
Alan Sussman
Elghazali Talbi
Jordi Torres
Cho-Li Wang
Adam Wierzbicki
Laurence T. Yang
Albert Zomaya

IS 2007 Program Committee

J.H. Abbawajy
André Adelsbach
Emmanuelle Anceaume
José Carlos Bacelar
Manuel Bernardo Barbosa
João Barros
Carlo Blundo
Phillip G. Bradford
Thierry Brouard
Han-Chieh Chao
Hsiao-Hwa Chen
Ilyoung Chong
Stelvio Cimato
Nathan Clarke
Miguel P. Correia
Cas Cremers
Gwenaël Doërr
Paul Dowland
Mahmoud T. El-Hadidi
Huirong Fu
Steven Furnell
Michael Gertz
Swapna S. Gokhale
Vesna Hassler
Lech J. Janczewski
Wipul Jayawickrama
Vasilis Katos
Hyun-KooK Kahng
Hiroaki Kikuchi
Paris Kitsos

Kwok-Yan Lam
Deok-Gyu Lee
Sérgio Tenreiro de Magalhães
Henrique S. Mamede
Evangelos Markatos
Arnaldo Martins
Paulo Mateus
Sjouke Mauw
Natalie Miloslavskaya
Edmundo Monteiro
Yi Mu
José Luís Oliveira
Nuno Ferreira Neves
Maria Papadaki
Manuela Pereira
Hartmut Pohl
Christian Rechberger
Carlos Ribeiro
Vincent Rijmen
José Ruela
Henrique Santos
Biplab K. Sarker
Ryoichi Sasaki
Jörg Schwenk
Paulo Simões
Filipe de Sá Soares
Basie von Solms
Stephanie Teufel
Luis Javier Garcia Villalba
Umberto Villano
Jozef Vyskoc
Carlos Becker Westphall
Liudong Xing
Chao-Tung Yang
Jeong Hyun Yi
Wang Yufeng
Deqing Zou
André Zũquete

ODBASE 2007 Program Committee

Andreas Abecker
Harith Alani
Jürgen Angele
Franz Baader
Sonia Bergamaschi
Alex Borgida
Mohand Boughanem
Paolo Bouquet
Jean-Pierre Bourey
Christoph Bussler
Silvana Castano
Paolo Ceravolo
Vassilis Christophides
Philipp Cimiano
Oscar Corcho
Ernesto Damiani
Ling Feng
Asuncion Gómez-Pérez
Benjamin Habegger
Mounira Harzallah
Andreas Hotho
Farookh Hussain
Dimitris Karagiannis
Manolis Koubarakis
Georg Lausen
Maurizio Lenzerini
Alexander Löser
Gregoris Metzas
Riichiro Mizoguchi
Boris Motik
John Mylopoulos
Wolfgang Nejdl
Eric Neuhold
Yves Pigneur
Axel Polleres
Li Qing
Wenny Rahayu
Rajugan Rajagopalapillai
Rainer Ruggaber
Heiko Schuldt
Eva Soderstrom
Wolf Siberski

Sergej Sizov
Chantal Soule-Dupuy
Umberto Straccia
Heiner Stuckenschmidt
VS Subrahmanian
York Sure
Francesco Taglino
Robert Tolksdorf
Guido Vetere
Roberto Zicari

OTM Conferences 2007 Additional Reviewers

Baptiste Alcalde
Soeren Auer
Abdul Babar
Luis Manuel Vilches Blázquez
Ralph Bobrik
Ngoc (Betty) Bao Bui
David Buján Carballal
Nuno Carvalho
Gabriella Castelli
Carlos Viegas Damasio
Jörg Domaschka
Viktor S. W. Eide
Michael Erdmann
Abdelkarim Erradi
Peter Fankhauser
Alfio Ferrara
Fernando Castor Filho
Ganna Frankova
Peng Han
Alexander Hornung
Hugo Jonker
Rüdiger Kapitz
Alexander Lazovik
Thortsen Liebig
Joäo Leitäo
Baochuan Lu
Giuliano Mega
Paolo Merialdo
Patrick S. Merten
Maja Milicic
Dominic Müller
Linh Ngo
José Manuel
Gómez Pérez
Sasa Radomirovic
Hans P. Reiser
Thomas Risse
Kurt Rohloff
Romain Rouvoy
Bernhard Schiemann
Jan Schlüter
Martin Steinert
Patrick Stiefel
Boris Villazón Terrazas
Hagen Voelzer
Jochem Vonk
Franklin Webber
Wei Xing
Christian Zimmer

Table of Contents – Part I

CoopIS 2007 International Conference (International Conference on Cooperative Information Systems)

CoopIS 2007 PC Co-chairs' Message

Welcome to the proceeding of the 15th International Conference on Cooperative Information Systems (CoopIS 2007) held in Vilamoura, Portugal, November 28-30, 2007.

The CoopIS conferences provide a forum for exchanging ideas and results on scientific research from a variety of areas, such as CSCW, Internet data management, electronic commerce, human–computer interaction, business process management, agent technologies, P2P systems, and software architectures, to name but a few. We encourage the participation of both researchers and practitioners in order to facilitate exchange and cross-fertilization of ideas and to support the transfer of knowledge to research projects and products. Towards this goal, we accepted both research and experience papers.

This year's conference included sessions that covered a broad range of topics in the design and development of cooperative information systems: process analysis and semantics, process modeling, P2P, collaboration, and business transactions.

This high-quality program would not have been possible without the authors who chose CoopIS as a venue to submit their publications. Out of 92 submitted papers, we selected 21 full papers and 7 short papers in a careful review process. Each submitted paper received at least three independent reviews. We are grateful for the dedicated work of the 58 experts in the field (including their co-reviewers) who served on the Program Committee and who did an excellent job.

To round up this excellent program, Microsoft Technical Fellow Donald Ferguson agreed to be our keynote speaker.

Finally, we are deeply indebted to Kwong Yuen Lai for his hard work, enthusiasm and almost constant availability. He was key to facilitating the paper management process and making sure that the review process stayed on schedule. We would also like to thank Robert Meersman and Zahir Tari for their support in organizing this event.

We hope that you enjoy this year's selection of contributions for CoopIS.

August 2007

Francisco Curbera
Frank Leymann
Mathias Weske

The Internet Service Bus

Donald F. Ferguson

Abstract. Service oriented architectures (SOA) increasingly form the basis for cooperative information systems. Many enterprises and application solutions have been practicing SOA for many years. SOA is an architecture that IT practitioners can layer on many technologies: messaging, RPC, etc. Most SOA solutions are evolving to exploit an "enterprise service bus" for integrating and connecting applications.

Web services are a set of standards that simplifies building cooperative information systems. The standards eliminate the need to enable the infrastructure to cooperate before enabling application cooperation.

Web services exploitation of Internet protocols and concepts creates an additional opportunity. Many, if not most, cooperating information systems have elements in different organizations: enterprises, sites, lines-of-business, etc. The same architectural style that led to an enterprise services bus will yield an Internet Service Bus.

This presentation will explain the Internet Service Bus concepts using a scenario. The scenario will also highlight many of the benefits of an ISB. There are several interesting technical and architectural challenges for building and using an ISB, which the presentation will discuss.

Speaker Bio

Dr. Donald F. Ferguson is a Microsoft Technical Fellow working in the Office of the CTO. Don focuses on advanced and future projects in the areas of enterprise computing and business applications. Before joining Microsoft, Don was an IBM Fellow and Chief Architect of IBM Software Group (SWG). He chaired the SWG Architecture Board, which provide overall technical guidance to WebSphere, Rational, Tivoli and Lotus products. Don was the founding technical lead for WebSphere. Don has contributed to several sets of standards, especially J2EE and Web services.

Soundness Verification of Business Processes Specified in the Pi-Calculus

Frank Puhlmann

Business Process Technology Group
Hasso Plattner Institut for IT Systems Engineering
University of Potsdam
D-14482 Potsdam, Germany
frank.puhlmann@hpi.uni-potsdam.de

Abstract. Recent research in the area of business process management (BPM) introduced the application of a process algebra—the π-calculus—for the formal description of business processes and interactions among them. Especially in the area of service-oriented architectures, the key architecture for today's BPM systems, the π-calculus—as well as other process algebras—have shown their benefits in representing dynamic topologies. What is missing, however, are investigations regarding the correctness, i.e. soundness, of process algebraic formalizations of business processes. Due to the fact that most existing soundness properties are given for Petri nets, these cannot be applied. This paper closes the gap by giving characterizations of invariants on the behavior of business processes in terms of bisimulation equivalence. Since bisimulation equivalence is a well known concept in the world of process algebras, the characterizations can directly be applied to π-calculus formalizations of business processes. In particular, we investigate the characterization of five major soundness properties, i.e. easy, lazy, weak, relaxed, and classical soundness.

1 Introduction

Process algebras, which are algebraic frameworks for the study of concurrent processes, recently gained extended attention in the area of business process management (BPM), e.g. [1,2,3,4,5,6,7]. This is especially true within the area of service-oriented architecture (SOA) [8], which is today's standard architectural style for realizing BPM solutions [9,10]. What the existing approaches lack, however, is a distinguished investigation on correctness properties for the business processes they describe. By correctness properties, we refer to the different kinds of soundness that have been introduced in the workflow management domain and later on refined. In particular, these are the original soundness definition by van der Aalst [11], nowadays denoted as classical soundness, relaxed soundness by Dehnert [12], and weak soundness by Martens [13]. Noteworthy, all these properties cannot directly be applied to process algebraic formalizations of business processes, since they are characterized using Petri nets. Furthermore, liveness and boundedness are used to prove business processes formalized with Petri nets to be sound; both techniques which are not available for process algebraic verification.

Table 1. Comparison of the different kinds of soundness

	Easy	Lazy	Weak	Relaxed	Classical
Possibility of termination	+	+	+	+	+
Support for lazy activities	+	+	-	+	-
Deadlock freedom	-	+	+	-	+
Participation of all activities	-	-	-	+	+

The focus of our research is on the application of a special kind of process algebra—the π-calculus—to the domain of business process management [14,15]. This calculus is of special interest, since it supports a direct representation of dynamic binding as found in service-oriented architectures [16]. In this paper, however, we do not tackle dynamic binding but instead investigate the correctness of the "inner workings" of the different services found in a SOA. The behavior of the "inner workings" of a service is described as a business process composed out of common patterns [17]. During our research on the formal representation of these patterns, we developed a new soundness property, that proves a business process to be lazy sound if it always provides a result [18]. While lazy soundness provides one way of proving an invariant of a business process represented in a process algebraic formalization, there exists no investigation that discusses the verification of business processes according to existing soundness properties. In a practical setting, however, different properties might be required. A comparison of the different kinds of soundness is shown in table 1.

Since the availability of correctness properties is a fundamental constraint for any formalization of business processes [19], we close the gap for π-calculus mappings by providing means to characterize soundness using bisimulation equivalence. Stated simply, a bisimulation is an equivalence relation between two processes, where the actions of both processes are matched, i.e. if one process can do an action, there exists a matching action in the other process and vice versa. Beyond providing characterizations for existing soundness properties, we discuss the declaration and reasoning on arbitrary invariants for the behavior of business processes using bisimulation equivalence. The discussion provides the reader with the theoretical equipment to his or her tailored soundness property.

The paper is organized as follows. We start with a short introduction to the π-calculus in section 2, followed by a discussion of how invariants for the behavior of business processes can be represented and proved. Section 3 provides characterizations of five major soundness properties in the π-calculus. The practical applicability of bisimulation equivalence is shown in section 4. Finally, we conclude with a discussion of related work in section 5.

2 Preliminaries

We start with an introduction to the π-calculus and introduce how business processes can be formalized using this algebra. Thereafter we discuss the representation of weak invariants, i.e. properties that have to be fulfilled by some instances of a business process, and strong invariants, i.e. properties that have to be fulfilled by all instances.

2.1 The Pi-Calculus

The π-calculus is a process algebra for the formal description and analysis of concurrent, interacting processes, denoted as agents. The calculus is based on names, that represent the unification of channels and data, used by agents defined according to [20].

Definition 1 (Pi-Calculus). *The syntax of the π-calculus is given by:*

$$\begin{aligned} P &::= M \mid P|P \mid \nu z\ P \mid A(y_1, \ldots, y_n) \\ M &::= \mathbf{0} \mid \pi.P \mid M + M \\ \pi &::= \overline{x}\langle \tilde{y} \rangle \mid x(\tilde{z}) \mid \tau \mid [x = y]\pi\ . \end{aligned}$$

P and M denote the agents and summations of the calculus. The informal semantics is as follows: $P|P$ is the concurrent execution of P and P, $\nu z\ P$ is the restriction of the scope of the name z to P, i.e. z is only visible in P and distinct from all other names, and $A(y_1, \cdots, y_n)$ denotes parametric recursion over the set of agent identifiers. $\mathbf{0}$ is inaction, a process that can do nothing, and $M+M$ is the exclusive choice between M and M. The prefixes of the calculus are given by π. The output prefix $\overline{x}\langle \tilde{y} \rangle.P$ sends a tuple of names $\tilde{y}$ via the co-name $\overline{x}$ and then continues as P. The input prefix $x(\tilde{z})$ receives a tuple of names via the name x and then continues as P with $\tilde{z}$ replaced by the received names. Matching input and output prefixes of different components might communicate, leading to an intraaction that is unobservable (τ). An explicit representation of an unobservable action is given by the prefix $\tau.P$ and the match prefix $[x = y]\pi.P$ behaves as $\pi.P$ if x is equal to y. Throughout this paper, upper case letters are used for agent identifiers and lower case letters for names. We abbreviate a set of components as $\prod_{i=1}^{n} Pi$, i.e. $\prod_{i=1}^{3} Pi = P_1 \mid P_2 \mid P_3$.

2.2 Business Process Formalizations

Informally, a business process can be seen as a a special process that creates a value or a result for a customer. A business process is characterized by activities and control flow relations between them. Additional ingredients are the types of the activities, if they route the control flow, or additional attributes, mostly related to control flow decisions. For reasoning on soundness, we abstract from other perspectives. Formally, a business process can be represented by a process graph given by:

Definition 2 (Process Graph). *A process graph is a four-tuple consisting of nodes, directed edges, types, and attributes. Formally: $P = (N, E, T, A)$ with*

- *N as a finite, non-empty set of nodes,*
- *$E \subseteq (N \times N)$ as a set of directed edges between nodes,*

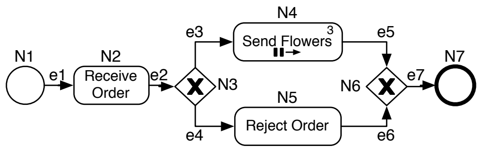


Fig. 1. A sample business process in BPMN notation

– *$T : N \rightarrow TYPE$ as a function from nodes to types, and*
– *$A \subseteq (N \times (KEY \times VALUE))$ as a relation from nodes to key/value pairs.*$\square$

N represents activities, incl. those responsible for routing the control flow given by E. T relates nodes with workflow patterns [21], where we assume simple activities or tasks to match the sequence pattern. A maps additional key/value pairs to nodes. Notable, a process graph only describes the static structure, i.e. the schema, of a business process. A process graph can easily be related to graphical notations such as EPCs [22], UML activity diagrams [23], or BPMN [24]. To give a process algebraic semantics to a process graph, the following, sketched algorithm is used (details can be found in [18]):

Algorithm 1 (Mapping Process Graphs to Agents). A process graph $P = (P_N, P_E, P_T, P_A)$ is mapped to π-calculus agents as follows:

1. All nodes of P are assigned a unique π-calculus agent identifier $N1 \ldots N|P_N|$.
2. All edges of P are assigned a unique π-calculus name $e1 \ldots e|P_E|$.
3. The π-calculus agents are defined according to the process patterns found in [17]. The functional perspective is represented by $\langle\cdot\rangle$. If the process graph is cyclic, recursion has to be used to allow multiple instances of activities.
4. An agent $N \overset{def}{=} (\nu e1, \ldots, e|P_E|\,)(\prod_{i=1}^{|P_N|} Ni)$ representing a process instance is defined. This agent might contain further components or restricted names according to the contained patterns. $\square$

The given mapping of a process graph represents a single instance (case) of a business process. During the evolution of the agent terms, their structure is reduced to future states, whereas all (possible) past states are lost. We showcase the formalization of an example given in figure 1, where we provide a graphical representation of the process graph using BPMN. Contained is a simple business process of a flower shipper. The shipper receives an order and either accepts the order, and sends flowers to three participants given in the order, or the order is rejected. The flowers are sent asynchronously via a multiple instance task without synchronization (denoted by the arrow at the bottom of the activity).

The formalization of the business process starts by assigning unique agent identifiers and names to the nodes and edges. These are already shown in the figure. The corresponding agents are given according to step three of algorithm 1 as follows:

$$N1 \overset{def}{=} \langle\cdot\rangle.\overline{e1}.\mathbf{0} \text{ and } N7 \overset{def}{=} e7.\langle\cdot\rangle.\mathbf{0}$$

represent the start and the end event. The functional perspective is abstracted by $\langle\cdot\rangle$, which will be filled later on. The nodes of the type task are given by the sequence workflow pattern:

$$N2 \overset{def}{=} e1.\langle\cdot\rangle.\overline{e2}.\mathbf{0} \text{ and } N5 \overset{def}{=} e4.\langle\cdot\rangle.\overline{e6}.\mathbf{0}\ .$$

The exclusive split and join gateways are given by

$$N3 \overset{def}{=} e2.\langle\cdot\rangle.(\overline{e3}.\mathbf{0} + \overline{e4}.\mathbf{0}) \text{ and } N6 \overset{def}{=} e5.\langle\cdot\rangle.\overline{e7}.\mathbf{0} + e6.\langle\cdot\rangle.\overline{e7}.\mathbf{0}\ .$$

The multiple instance task with three static instances is given by

$$N4 \overset{def}{=} e3.(\langle\cdot\rangle.\mathbf{0} \mid \langle\cdot\rangle.\mathbf{0} \mid \langle\cdot\rangle.\mathbf{0} \mid \overline{e5}.\mathbf{0})\ .$$

Finally, an agent

$$N \overset{def}{=} (\nu e1,\ldots,e7\,)(\prod_{i=1}^{7} Ni)\ .$$

represents a fresh process instance. An extended discussion of the semantics and the mapping can be found in [17,18].

2.3 Simulation

To specify that a process instance given by π-calculus agents *can* fulfill an invariant, we need the concept of simulation from the process algebraic toolbox. Informally, a simulation relates an agent P with another agent Q, if Q can follow every action of P. We say that Q can simulate P. The actions of the agents are given by the input, output, and unobservable prefixes, denoted as $Act = \{\overline{x}\langle\tilde{y}\rangle, x(\tilde{z}), \tau\}$. The evolution of the state of an agent to a succeeding state is denoted by a transition bearing the corresponding action, i.e. $\overline{a}\langle w\rangle.A \xrightarrow{\overline{a}\langle w\rangle} A$.

Definition 3 (Simulation). *A simulation is a binary relation $\mathcal{R}$ on agents such that $\forall \alpha \in Act$:*

$$P\mathcal{R}Q \wedge P \xrightarrow{\alpha} P' \Rightarrow \exists Q' : Q \xrightarrow{\alpha} Q' \wedge P'\mathcal{R}Q'\ .$$

Q is similar to P, denoted as $P \precsim Q$, if they are related by a simulation.

Simulation considers a strong relation between interactions and unobservable actions. Two agents

$$P \overset{def}{=} a(x).\tau.\tau.\overline{b}\langle z\rangle.\mathbf{0} \text{ and } Q \overset{def}{=} a(x).\tau.\overline{b}\langle z\rangle.\mathbf{0}$$

cannot simulate each other, since they differ in the number of their unobservable actions (τ transitions). A simulation that abstracts from these unobservable actions is called weak simulation. Weak simulations are of particular interest, since they abstract from the internal behavior of agents and instead only consider the external visible behavior. A weak simulation is obtained by defining $\Longrightarrow$ to represent zero or more τ transitions, i.e. $\xrightarrow{\tau}^{*}$, $\overset{\alpha}{\Longrightarrow}$ as $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$, and $\overset{\hat{\alpha}}{\Longrightarrow}$ as $\overset{\alpha}{\Longrightarrow}$ if $\alpha \neq \tau$ and $\Longrightarrow$ if $\alpha = \tau$.

Definition 4 (Weak Simulation). *A weak simulation is a binary relation $\mathcal{R}$ on agents such that $\forall \alpha \in Act$:*

$$P \mathcal{R} Q \wedge P \xrightarrow{\alpha} P' \Rightarrow \exists Q' : Q \overset{\hat{\alpha}}{\Longrightarrow} Q' \wedge P' \mathcal{R} Q' \ .$$

Q is weak similar to P, denoted as $P \precapprox Q$, if they are related by a weak simulation.

By using weak simulation, we can match π-calculus mappings of process graphs for the potential fulfillment of arbitrary invariants. Therefore we specify the invariant by a π-calculus agent that trivially fulfills them. Consider for instance

$$I1 \overset{def}{=} \overline{s}.\mathbf{0} \ ,$$

that denotes that an activity can occur during the execution of a process instance via an emission of $\overline{s}$. To enhance the agent that represents the activity under investigation in the π-calculus mapping, we need to replace $\langle \cdot \rangle$ by $\overline{s}$. Let's assume we want to prove that the reject order task from figure 1 can be executed. Hence, we change agent $N5$ accordingly:

$$N5 \overset{def}{=} e4.\overline{s}.\overline{e6}.\mathbf{0} \ .$$

We assume all other functional abstractions to be filled with unobservable actions (τ). Now we can decide if $I1 \precapprox N$ holds by finding a relation $\mathcal{R}$ that relates both agents. If we are able to find such a relation, we proved that the reject order task can indeed be executed. If we found a counterexample, i.e. something that $I1$ can do but N is unable to simulate, we disproved the invariant for the business process. The magic of weak simulation lies in the fact that all internal interactions between components, i.e. between $N1$ and $N2$ via $e1$ are unobservable from the outside, meaning they resemble τ. Since all names $e*$ are restricted inside N, only $\overline{s}$ has to be considered in the simulation. Regarding this semantics, the agent N breaks down to a number of τ actions, a possible emission via $\overline{s}$, and another number of τ actions. Hence, it resembles $\overset{\hat{\overline{s}}}{\Longrightarrow}$. According to definition 4, $\mathcal{R}$ is given by $\{(I1, N), (\mathbf{0}, \mathbf{0})\}$ and $I1 \precapprox N$ holds. Thus, the business process from figure 1 fulfills the invariant that the reject order task can be executed.

We can also disprove invariants for business processes, i.e. show that the reject order activity will never be executed twice per instance. Therefore we modify the agent representing the invariant to

$$I2 \overset{def}{=} \overline{s}.\overline{s}.\mathbf{0} \ .$$

Due to the fact that $I2 \xrightarrow{\overline{s}} \overline{s}.\mathbf{0}$, and correspondingly $N \overset{\hat{\overline{s}}}{\Longrightarrow} \mathbf{0}$, the remainder of $I2$ can do another action $\overline{s}$ that the remainder $\mathbf{0}$ of N cannot simulate. Hence, $I2 \not\precapprox N$ disproves the supposed invariant.

2.4 Bisimulation

By using weak simulation we are able to show an optional behavior. This is due to the fact that a simulation only investigates one direction. If the agent mapping of a process graph contains additional actions, it is not investigated if the agent representing the invariant can mimic them. To enforce that two agents are able to mimic all their actions in arbitrary directions, i.e. the first agent does something, the second agent corresponds, and thereafter the second agent does something else that the first agent needs to mimic, etc., we require the relation $\mathcal{R}$ to be symmetric. Helpful is again the weak variant, yielding weak bisimulation:

Definition 5 (Weak Bisimulation). *A weak bisimulation is a symmetric, binary relation $\mathcal{R}$ on agents such that $\forall \alpha \in Act$:*

$$P\mathcal{R}Q \wedge P \xrightarrow{\alpha} P' \Rightarrow \exists Q' : Q \overset{\hat{\alpha}}{\Longrightarrow} \alpha Q' \wedge P'\mathcal{R}Q' \ .$$

P and Q are weak bisimilar, denoted as $P \approx Q$, if they are related by a weak bisimulation.

According to this definition, weak bisimulation equivalence, or weak bisimilarity, between two agents P and Q is stronger than mutual simulation, i.e. from $P \approx Q \Rightarrow P \precapprox Q \wedge Q \precapprox P$, but the converse does not necessarily hold. Consider for instance once again $I1$ and N as given in the previous section. While $I1 \precapprox N$ holds (as shown), also $N \precapprox I1$ holds (proof left for the reader). However, this does not mean that the reject order task is executed in every instance, as can be easily checked in figure 1. The technical difference lies in the fact that bisimulation is symmetric, a property that allows switching the direction after every step instead assuming a fixed order. According to bisimulation, we need to find a counterexample to prove $I1 \not\approx N$ and thereby disprove the proposition that reject order is executed in every instance. Since this is a quite complex task, we refer the reader to section 4, where we introduce tool-supported reasoning (indeed, a counterexample can be found).

The last application of bisimulation that we will consider is proving invariants that hold for all instances. Regarding figure 1, it seems obvious that receive order is executed in every instance. We can prove this proposition by returning to the original definition of N and modify its component $N2$, representing the receive order task, accordingly:

$$N2 \overset{def}{=} e1.\overline{s}.\overline{e2}.\mathbf{0} \ .$$

By finding a relation $\mathcal{R}$ between $I1$ and the modified N according to definition 5 we can prove that receive order is executed in each instance. Since such an relation exists, $I1 \approx N$ holds (for the modified N). As the relation contains 29 tuples, we once again refer to section 4 for tool-supported reasoning.

3 Characterizations of Soundness

After having shown how *can* properties of business processes are proved using simulation and *must* properties are proved using bisimulation, we discuss existing soundness properties. Since most existing properties are given for workflow nets [25], a subclass of Petri nets, we define a subset of process graphs that fulfills the same structural properties. We denote this property as *structural soundness.* Informally, structural soundness is given by:

> A process graph is structural sound if it has exactly one initial node, exactly one final node, and all other nodes lie on a path between the initial and the final node.

Structural sound process graphs resemble placeholders for business processes with the denoted structural properties. We omit the (obvious) formal definition due to space limits.

3.1 Easy Soundness

The least soundness property a business process should fulfill is given by *easy soundness*, informally given by:

> A structural sound process graph representing a business process is easy sound if a result can be provided.

As indicated by the word *can*, we have to use simulation to prove this property for process algebraic formalizations of business processes. In particular, we have to be able to observe the occurrence of the initial and the final node. The idea is depicted in figure 2. A structural sound process graph is fed into a black box. Each time we press the start button, an instance of the process graph is executed. Each time the final node of process graph is executed, the done bulb flashes. Regarding easy soundness, we have to find at least one process instance where the done bulb flashes, denoting the delivery of the result. An agent fulfilling this invariant is given by:

$$S_{EASY} \overset{def}{=} i.\tau.\overline{o}.\mathbf{0}\ . \tag{1}$$

The input prefix i denotes the pushbutton, whereas the output prefix $\overline{o}$ resembles the done bulb. Both are in a fixed sequence, i.e. $\overline{o}$ follows always after i. The τ in-between denotes the abstraction from complex actions. Since we use weak (bi)-simulations, however, it could also be omitted. To be able to decide whether a business process given by π-calculus agents is weak similar to S_{EASY}, we have to enhance the agents representing the business process:

Algorithm 2 (Easy Soundness Annotated Pi-Calculus Mapping). The π-calculus mapping of a process graph according to algorithm 1 is enhanced for reasoning on easy soundness as follows. The functional abstraction $\langle\cdot\rangle$ of (1) the agent that represents the initial node is replaced by $i.\tau$; (2) the agent that represents the final node is replaced by $\tau.\overline{o}$; (3) all other agents are replaced by τ. Obviously, i and o are not permitted to appear anywhere else in the agent terms. □

A formal definition of easy soundness using weak similarity is now given by:

Definition 6 (Easy Sound Process Graph). *A structural sound process graph P with a semantics given by the easy soundness annotated π-calculus mapping D of P is easy sound if $S_{EASY} \lessapprox D$ holds.*

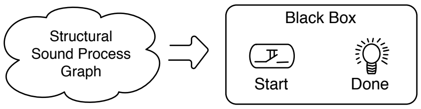


Fig. 2. Black box investigation of a structural sound process graph

We can prove the sample business process from figure 1 to be easy sound by finding a relation for $S_{EASY} \precapprox N_{EASY}$, with N_{EASY} being syntactically equal to N with $\langle\cdot\rangle$ replaced by τ and:

$$N1 \overset{def}{=} i.\tau.\overline{e1}.\mathbf{0} \text{ and } N7 \overset{def}{=} e7.\tau.\overline{o}.\mathbf{0} .$$

Since such a relation exists (4 tuples), the business process of the flower shipper is easy sound. The relation can be reconstructed by the reader as will be shown in section 4.

3.2 Lazy Soundness

One obvious extension to easy soundness is given by enforcing that all instances of a process graph provide a result:

> A structural sound process graph representing a business process is lazy sound if in any case a result is provided exactly once.

This property can be proved using weak bisimilarity. Furthermore, all assumptions from easy soundness also hold. In particular, we need to be able to observe the occurrence of the final node after *each* occurrence of the initial node. Regarding the black box from figure 2, this means that each time the start button is pressed, we need to be able to observe exactly one flash of the done bulb. In contrast to easy soundness, we cannot try until we observe a flash of the done bulb, but have to consider all possibilities instead. This supplies two problems: (1) How can we be sure that all path of the process graph have been traversed, i.e. we do not need to press the start button anymore; (2) How do we know if we do not need to wait any longer for the done bulb to flash, i.e. a deadlock has occurred? If we are able to find a bisimulation between an invariant given by

$$S_{LAZY} \overset{def}{=} i.\tau.\overline{o}.\mathbf{0} \tag{2}$$

and an annotated agent mapping of a process graph, both problems have been overcome. The former due to the fact that a bisimulation enumerates all possible states and the latter by the fact that a bisimulation is finite. Since S_{LAZY} exactly resembles S_{EASY}, the same annotation for the agents has to be used:

Algorithm 3 (Lazy Soundness Annotated Pi-Calculus Mapping). The same as given by algorithm 2. □

A formal definition of lazy soundness using weak bisimilarity is now given by:

Definition 7 (Lazy Sound Process Graph). *A structural sound process graph P with a semantics given by the lazy soundness annotated π-calculus mapping D of P is lazy sound if $D \approx S_{LAZY}$ holds.*

The business process from figure 1 can be checked for satisfying lazy soundness; i.e. we need to prove that $S_{LAZY} \approx N_{EASY}$ holds.

3.3 Weak Soundness

Lazy soundness only considers the return of a result, whereas the termination of the process graph, i.e. all nodes are terminated, is not considered. This is due to the fact that deferred, so called *lazy*, activities can remain active after the result has been provided (an extended discussion can be found in [18]). Nevertheless, in some cases it has to be guaranteed that the termination of a business process occurs the very moment the result is provided:

> A structural sound process graph representing a business process is weak sound if in any case a result is provided and the process instance is terminated the moment the result is provided.

Since once again all cases need to be considered, weak bisimilarity is the technique of choice. What has to be changed, however, is the black box we use for investigation. In addition to be able to observe the occurrence of the initial and the final node, we also need to be able to observe the occurrence of each other node. This enhancement is depicted in figure 3. A business process placed inside the enhanced black box is weak sound if we are unable to observe a flash of the step bulb after a flash of the done bulb. Furthermore, a flash of the done bulb has to be observed exactly once for each push on the start button. Deriving the invariant is not this easy, however. A naive version given by

$$I \overset{def}{=} i.I1 \text{ and } I1 \overset{def}{=} \overline{s}.I1 + \tau.\overline{o}.\mathbf{0}$$

will not work with an annotated π-calculus mapping given below (the proof is left to the reader; hint: $I1$ can send an unlimited times via $\overline{s}$). The problem can be overcome by allowing each instance of a process graph to emit via $\overline{s}$ only once. The choice which node will emit via $\overline{s}$ has to be made *non-deterministically.* This is done via an activity observation agent that will be included in the π-calculus mapping of a process graph later on:

$$\begin{aligned} X(x,s) &\overset{def}{=} x(ack).(\tau.\overline{ack}.\mathbf{0} \mid X(x,s)) + x(ack).(\tau.\overline{s}.\overline{ack}.\mathbf{0} \mid X_1(x)) \\ X_1(x) &\overset{def}{=} x(ack).(\tau.\overline{ack}.\mathbf{0} \mid X_1(x)) \ . \end{aligned} \tag{3}$$

Agent X is triggered via x and thereafter has the non-deterministic choice between acknowledging via *ack* or emitting via $\overline{s}$ and thereafter acknowledging. If the former happened, X behaves again as X. If the latter happened, X behaves as $X1$ that is only capable of acknowledging. Due the activity observation agent,

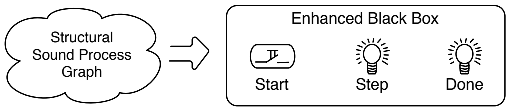


Fig. 3. Enhanced black box investigation of a structural sound process graph

each node of a process graph has the capability of signaling its execution. As this explicitly includes nodes of a process graph that are active after the final node has signaled its execution, weak soundness can be proved by an invariant given as

$$S_{WEAK} \stackrel{def}{=} i.(\tau.\overline{o}.\mathbf{0} + \tau.\overline{s}.\overline{o}.\mathbf{0}) \ . \tag{4}$$

S_{WEAK} is the same as S_{LAZY} regarding i and $\overline{o}$. After the observation of the initial node via i, a choice between observing $\overline{o}$ or $\overline{s}$ is made. If $\overline{o}$ is observed, no other observations are possible (due to S_{WEAK} becomes inaction). If $\overline{s}$ is observed, the next observation has to be $\overline{o}$. Thereafter, no other observations are possible. This behavior resembles the enhanced black box with the exception that the step bulb might flash only once before/after the done bulb flashed. The agents that represent the business process have to be enhanced as given by the following algorithm:

Algorithm 4 (Weak Soundness Annotated Pi-Calculus Mapping). The π-calculus mapping D of a process graph $P = (N, E, T, A)$ according to algorithm 1 is enhanced for reasoning on weak soundness as follows. The functional abstraction $\langle\cdot\rangle$ of (1) the agent that represents the initial node is replaced by $\nu ack\ i.\overline{x}\langle ack\rangle.ack$; (2) the agent that represents the final node is replaced by $\nu ack\ \overline{x}\langle ack\rangle.ack.\overline{o}.\tau$; (3) all other agents are replaced by $\nu ack\ \overline{x}\langle ack\rangle.ack$. Furthermore, the agent from equation 3 has to be included in D:

$$D \stackrel{def}{=} (\nu e1, \ldots, e|E|, x)(\prod_{i=1}^{|N|}(Di) \mid X) \ .$$

The names i, o, and s are not permitted to appear anywhere else in the agent terms. □

A formal definition of weak soundness for a process graph is now given by:

Definition 8 (Weak Sound Process Graph). *A structural sound process graph P with a semantics given by the weak soundness annotated π-calculus mapping D of P is weak sound if $D \approx S_{WEAK}$ holds.*

The business process from figure 1 is not fulfilling weak soundness. This is due to the fact that the flowers are sent asynchronously; i.e. the send flowers activity is lazy.

3.4 Relaxed Soundness

All preceding soundness properties neglect an investigation regarding the participation of activities in a business process. Sometimes this is an important property, due to the fact that unused activities can be removed from a business process. Similar to [12], our interpretation of *relaxed soundness* also supports the synchronizing merge pattern:

> A structural sound process graph representing a business process is relaxed sound if each node of the process graph has the possibility of being executed in between the execution of the initial and the final node.

Since we talk about a possibility, weak similarity has to be used this time. We can reuse the enhanced black box from figure 3 with a special preparation of the π-calculus mapping. In particular, we need to prepare as much copies of the π-calculus mapping as there are nodes in the process graph. In each copy we need to give another node the possibility of emitting via $\overline{s}$ to signal its execution. Hence, for each enhanced mapping of a process graph feed into the enhanced black box, we should be able to observe a flash of the step and done bulb in sequence in at least one pass. The invariant is given by:

$$S_{RELAXED} \overset{def}{=} i.\overline{s}.\overline{o}.\mathbf{0} \ .$$

The agent $S_{RELAXED}$ simply resembles the execution order. However, special care has to be taken if the node under observation is always executed more than once (the next action of $S_{RELAXED}$ after $\overline{s}$, $\overline{o}$, cannot be simulated by a mapping that emits $\overline{s}$ once again!). This problem can be solved by including an activity loop observation agent that only emits $\overline{s}$ for the first execution of a node:

$$Y(y, s) \overset{def}{=} y(ack).\overline{s}.\overline{ack}.\,Y_1(y) \ \text{ and } \ Y_1(y) \overset{def}{=} y(ack).\tau.\overline{ack}.\,Y_1(y) \ .$$

The agent is integrated in the mapping of a process graph as follows:

Algorithm 5 (Relaxed Soundness Annotated Pi-Calculus Mapping). To annotate a π-calculus mapping D of a process graph $P = (N, E, T, A)$ for a certain node $n \in N \setminus \{x, y\}$ with $T(x) = \mathit{InitialNode}, T(y) = \mathit{FinalNode}$ for reasoning on relaxed soundness regarding the node n, the following steps have to be made. The functional abstraction $\langle\cdot\rangle$ of (1) the agent that represents the initial node is replaced by $i.\tau$; (2) the agent that represents the final node is replaced by $\tau.\overline{o}$; (3) the agent that represents n is replaced by $\nu ack\ \overline{y}\langle ack\rangle.ack$; (4) all other agent is replaced by τ. Furthermore, the agent from equation 3.4 has to be included in D:

$$D \overset{def}{=} (\nu e1, \ldots, e|E|, y)(\prod_{i=1}^{|N|}(Di) \mid Y) \ .$$

The names i, o, and s are not permitted to appear anywhere else in the agent terms. □

Relaxed soundness is formally given by:

Definition 9 (Relaxed Sound Process Graph). *A structural sound process graph $P = (N, E, T, A)$ is relaxed sound if for each relaxed soundness annotated π-calculus mapping D considering $n \in N \setminus \{x, y\}$ with $T(x) = \mathit{InitialNode}, T(y) = \mathit{FinalNode}$ it holds that $S_{RELAXED} \precsim D$.*

Regarding the example from figure [1], it can be shown that each node has the possibility to participate in the business process by calculating the corresponding weak simulations.

3.5 Classical Soundness

A strong soundness property, known as (classical) soundness [11], is given informally by:

> A structural sound process graph representing a business process is sound if (1) in any case a result is provided; (2) the process instance is terminated the moment the result is provided; and (3) each node of the process graph has the possibility of being executed after the initial node.

The first two criteria coincidence with weak soundness. The last criterion is given by a modified version of relaxed soundness, where the activity loop observation agent and $\overline{o}$ are omitted from the π-calculus mapping. The relaxed invariant is given by:

$$S_{PART} \overset{def}{=} i.\overline{s}.\mathbf{0} \tag{5}$$

We can omit the activity loop observation agent due to the fact that multiple emissions via $\overline{s}$ from the π-calculus mapping of the process graph are not disturbing the similarity because $\overline{o}$ is omitted.

Algorithm 6 (Participating Annotated Pi-Calculus Mapping). To annotate a π-calculus mapping of a process graph $P = (N, E, T, A)$ for a certain node $n \in N \setminus \{x, y\}$ with $T(x) = \mathit{InitialNode}, T(y) = \mathit{FinalNode}$ for reasoning on the participation of n in the business process, the following steps have to be made. The functional abstraction $\langle\cdot\rangle$ of (1) the agent that represents the initial node is replaced by $i.\tau$; (2) the agent that represents the node n is replaced by $\overline{s}$; (3) all other agents is replaced by τ. The names i and s are not permitted to appear anywhere else in the agent terms. $\square$

The definition of classical soundness using weak similarity and bisimilarity is formally given by:

Definition 10 (Classical Sound Process Graph). *A structural sound process graph $P = (N, E, T, A)$ with a semantics given by (1) the weak soundness annotated π-calculus mapping D1 of P and (2) a set of participating annotated π-calculus mappings D2 for each $n \in N \setminus \{x, y\}$ with $T(x) = \mathit{InitialNode}, T(y) = \mathit{FinalNode}$ is classical sound if it holds that (a) $D1 \approx S_{WEAK}$ and (b) $S_{PART} \precapprox D$ for each $D \in D2$.*

4 Tool Support and Efforts

This first part of this section shows the practical applicability of the different soundness characterization using an existing tool. The second part discusses an important criterion: The efforts required for deciding bisimulation equivalence for the different kinds of soundness.

4.1 Tool-Supported Reasoning

Reasoning on similarity and bisimilarity of π-calculus agents can be done using the Advanced Bisimulation Checker (ABC).[1] The reasoner accepts agents in an ASCII syntax described in the corresponding documentation. In a nutshell, `'x<y>` represents an output prefix, `x(y)` an input prefix, `t` an unobservable action, and `(^z)` the restriction operator. Regarding the easy/lazy soundness annotated mapping from figure 1, the following input is appropriate:

```
agent N1(e1,i)=i.t.'e1.0
agent N2(e1,e2)=e1.t.'e2.0
agent N3(e2,e3,e4)=e2.t.('e3.0 + 'e4.0)
agent N4(e3,e5)=e3.(t.0 | t.0 | t.0 | 'e5.0)
agent N5(e4,e6)=e4.t.'e6.0
agent N6(e5,e6,e7)=e5.t.'e7.0 + e6.t.'e7.0
agent N7(e7,o)=e7.t.'o.0
agent N(i,o)=(^e1,e2,e3,e4,e5,e6,e7)(N1(e1,i) | N2(e1,e2) | N3(e2,e3,e4) | N4(e3,e5) |
      N5(e4,e6) | N6(e5,e6,e7) | N7(e7,o))
agent S_EASY(i,o)=i.t.'o.0
agent S_LAZY(i,o)=i.t.'o.0
```

Easy soundness can be decided by asking ABC for proving similarity between S_{EASY} and N using the `wlt` command:

```
abc > wlt S_EASY(i,o) N(i,o)
The two agents are weakly related (4).
```

Since a simulation exists, easy soundness for the business process from figure 1 has been proved. Lazy soundness can be decided by proving bisimilarity between S_{LAZY} and N using the `weq` command:

```
abc > weq S_LAZY(i,o) N(i,o)
The two agents are weakly related (70).
```

Both agents are bisimilar due to the fact that a bisimulation has been found. A session disproving weak soundness for the example is given by:

```
agent N1(e1,i,x)=i.(^ack)'x<ack>.ack.'e1.0
agent N2(e1,e2,x)=e1.(^ack)'x<ack>.ack.'e2.0
agent N3(e2,e3,e4,x)=e2.(^ack)'x<ack>.ack.('e3.0 + 'e4.0)
agent N4(e3,e5,x)=e3.((^ack)'x<ack>.ack.0 | (^ack)'x<ack>.ack.0 | (^ack)'x<ack>.ack.0 |'e5.0)
agent N5(e4,e6,x)=e4.(^ack)'x<ack>.ack.'e6.0
agent N6(e5,e6,e7,x)=e5.(^ack)'x<ack>.ack.'e7.0 + e6.(^ack)'x<ack>.ack.'e7.0
agent N7(e7,o,x)=e7.(^ack)'x<ack>.ack.'o.0
agent X(x,s)=x(ack).(t.'ack.0 | X(x,s)) + x(ack).('s.'ack.0 | X_1(x))
agent X_1(x)=x(ack).(t.'ack.0 | X_1(x))
agent N(i,o,s)=(^e1,e2,e3,e4,e5,e6,e7,x)(N1(e1,i,x) | N2(e1,e2,x) | N3(e2,e3,e4,x) |
      N4(e3,e5,x) | N5(e4,e6,x) | N6(e5,e6,e7,x) | N7(e7,o,x) | X(x,s))
agent S_WEAK(i,o,s)=i.(t.'o.0 + t.'s.'o.0)

abc > weq S_WEAK(i,o,s) N(i,o,s)
The two agents are not weakly related (30).
```

Further examples are omitted due to a lack of space.

[1] Available at `http://lampwww.epfl.ch/~sbriais/abc/abc.html`

4.2 Efforts

This subsection takes a closer look at the complexity of deciding bisimulation. In the general case, bisimulation equivalence on π-calculus agents is undecidable. This is due to the Turing-completeness of the calculus, e.g. shown in [20]. What can be decided, however, is non-equivalence of agents, since after finite number of transitions, a counterexample has to be found. Nevertheless, our aim is to prove that a π-calculus mapping of a business process fulfills a certain property, hence it is equivalent.

The problems can partly be overcome by restricting the grammar of the π-calculus variant applied. For the following discussion, we consider a business process with a number of nodes, given by the following agent:

$$N \overset{def}{=} (e1, e2, \dots)(\prod_{i=1} Ni) \ .$$

Agents with Simple Sequences. Simple sequences, such as

$$N1 \overset{def}{=} \langle\cdot\rangle.\overline{e1}.\mathbf{0} \ , \ \ N2 \overset{def}{=} e1.\langle\cdot\rangle.\overline{e2}.\mathbf{0} \ \text{ and } \ N3 \overset{def}{=} n2.\langle\cdot\rangle.\mathbf{0} \ ,$$

can be enforced by removing recursion via defined agent identifiers from the calculus. As a result, loops are prohibited. This significantly drops the effort for most practical problem sizes. However, we also loose Turing-completeness.

Agents Mappings with Loops in the Business Processes. Agents that represent business processes with loops, such as

$$N1 \overset{def}{=} \langle\cdot\rangle.\overline{e1}.\mathbf{0} \ , \ \ N2 \overset{def}{=} e1.\langle\cdot\rangle.((\overline{e1}.\mathbf{0} + \overline{e2}.\mathbf{0}) \mid N2) \ \text{ and } \ N3 \overset{def}{=} n2.\langle\cdot\rangle.\mathbf{0} \ ,$$

can in most cases efficiently be checked, because the same state(s) appears over and over again. However, we do not allow the creation of restricted names in recursive passages, since this would lead to the next problem class.

Arbitrary Recursion and Restrictions. Agents such as

$$\begin{aligned} A &\overset{def}{=} a.(A_1(b) \mid A_2) \\ A_1(prev) &\overset{def}{=} \nu next \ \overline{create\_i}\langle next, prev\rangle.A_1(next) + \overline{prev}.\mathbf{0} \\ A_2 &\overset{def}{=} create\_i(next, prev).(\langle\cdot\rangle.next.\overline{prev}.\mathbf{0} \mid A_2) \ . \end{aligned}$$

where arbitrary restricted names can be created in recursive passages are hard to verify, because new states are created all the way. However, this kind of problem is only to be found in the multiple instances workflow patterns (as shown), which can be abstracted by τ for verification.

Agents with massive non-determinism. Agents such as X and Y according to weak and relaxed soundness contain massive amounts of non-determinism. This has an exponential influence on the state space that needs to be checked. Consequently, the most promising property regarding computational complexity is lazy soundness.

Solutions. Our current efforts go into the direction of implementing a domain-specific bisimulation checker for BPM. The already restricted input set given by process graphs is further stripped down by applying the asynchronous π-calculus [20], which is also able to represent all workflow patterns. The goal of our research is not limiting the input further, e.g. by only allowing block structures or prohibiting loops. Instead, we are working on a simplification of the workflow pattern formalization, the normalization and optimization of the generated agent term, as well as including heuristics via external data (e.g. process graphs). In this paper, we laid the formal foundations behind bisimulation-based soundness verification.

5 Conclusion and Related Work

In this paper we have shown how invariants for π-calculus mappings of business processes can be declared and proved. Besides introducing the general concepts in section 2, we also investigated easy, lazy, weak, relaxed, and classical soundness in section 3. The practical feasibility of our findings has been sketched afterwards in section 4, where we sketched the question of computational complexity. However, future research in this area is crucial for the practical applicability. In particular, we will investigate different classes of inputs vs. different soundness properties. While weak, relaxed, and classical soundness rely on link passing mobility, that is not available in all process algebras, the general concepts can also be applied to other algebras like CCS [26]. We already presented a tool chain for lazy soundness as part of earlier research [18,27]. This paper goes one step further by discussing the general concepts as well as missing soundness properties. To the knowledge of the authors, no other approach using similarity and bisimilarity for deciding different kinds of soundness has been published. Nevertheless, as we already sketched in [27], also projection inheritance [28] for Petri nets can be used.

Regarding foundational work, the different soundness definitions from van der Aalst [11], Dehnert [12], and Martens [13] directly inspired our definitions. Since these are given for Petri nets, we could only informally resemble them. For instance, the black box verification of lazy soundness closely resembles the first criterion of soundness for workflow nets:

$$\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o) \; .$$

It states that a workflow net has the option to always complete, i.e. deliver a result from our perspective. The second criterion,

$$\forall_M (i \xrightarrow{*} M \wedge M \geq o) \Rightarrow (M = o) \; ,$$

is resembled by weak soundness and the enhanced black box observation. It states that a workflow net terminates the moment a token is in the final place, i.e. the result is provided the moment the process instance is terminated. The third criterion,

$$\forall_{t \in T} \exists_{M,M'} i \xrightarrow{*} M \xrightarrow{t} M' \; ,$$

states that each task of a workflow net can participate in the workflow. It is resembled by a subset of relaxed soundness as described in section 3.5.

Remarks. The definition of weak bisimulation has been simplified, since otherwise more elaborate foundations for the π-calculus would be required (e.g. bound and free names). The reader is refered to [14].

References

1. Brogi, A., Canal, C., Pimentel, E., Vallecillo, A.: Formalizing Web Service Choreographies. In: Proceedings of First International Workshop on Web Services and Formal Methods. Electronic Notes in Theoretical Computer Science, Elsevier, Amsterdam (2004)
2. Laneve, C., Zavattaro, G.: Foundations of Web Transactions. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 282–298. Springer, Heidelberg (2005)
3. Bordeaux, L., Salaün, G.: Using Process Algebra for Web Services: Early Results and Perspectives. In: Shan, M.-C., Dayal, U., Hsu, M. (eds.) TES 2004. LNCS, vol. 3324, pp. 54–68. Springer, Heidelberg (2005)
4. Guidi, C., Lucchi, R., Gorrieri, R., Busi, N., Zavattaro, G.: SOCK: A Calculus for Service Oriented Computing. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 327–338. Springer, Heidelberg (2006)
5. Mazzara, M., Lanese, I.: Towards a Unifying Theory for Web Service Composition. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 257–272. Springer, Heidelberg (2006)
6. Ferrara, A.: Web Services: A Process Algebra Approach. In: ICSOC 2004. Proceedings of the 2nd international conference on Service oriented computing, pp. 242–251. ACM Press, New York, NY, USA (2004)
7. Decker, G., Zaha, J.M., Dumas, M.: Execution Semantics for Service Choreographies. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 163–177. Springer, Heidelberg (2006)
8. Burbeck, S.: The Tao of E-Business Services (2000)
9. Woodley, T., Gagnon, S.: BPM and SOA: Synergies and Challenges. In: Ngu, A.H.H., Kitsuregawa, M., Neuhold, E.J., Chung, J.-Y., Sheng, Q.Z. (eds.) WISE 2005. LNCS, vol. 3806, pp. 679–688. Springer, Heidelberg (2005)
10. Newcomer, E., Lomov, G.: Understanding SOA with Web Services. Addison–Wesley, London (2005)
11. Aalst, W.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
12. Dehnert, J., Rittgen, P.: Relaxed Soundness of Business Processes. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 157–170. Springer, Heidelberg (2001)
13. Martens, A.: Analyzing Web Service based Business Processes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 19–33. Springer, Heidelberg (2005)
14. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part I/II. Information and Computation 100, 1–77 (1992)
15. Puhlmann, F.: Why do we actually need the Pi-Calculus for Business Process Management? In: Abramowicz, W., Mayr, H. (eds.) BIS 2006. 9th International Conference on Business Information Systems, Bonn. LNI, vol. P-85, pp. 77–89. Gesellschaft für Informatik (2006)

16. Overdick, H., Puhlmann, F., Weske, M.: Towards a Formal Model for Agile Service Discovery and Integration. In: Proceedings of the International Workshop on Dynamic Web Processes (DWP 2005). IBM technical report RC23822, Amsterdam (2005)
17. Puhlmann, F., Weske, M.: Using the Pi-Calculus for Formalizing Workflow Patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 153–168. Springer, Heidelberg (2005)
18. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, pp. 145–160. Springer, Heidelberg (2006)
19. Aalst, W., Hofstede, A., Weske, M.: Business Process Management: A Survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
20. Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. Paperback edn. Cambridge University Press, Cambridge (2003)
21. Aalst, W., Hofstede, A., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14, 5–51 (2003)
22. Keller, G., Nüttgens, M., Scheer, A.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Technical Report 89, Institut für Wirtschaftsinformatik, Saarbrücken (1992)
23. OMG: UML 2.0 Superstructure Final Adopted specification (2003)
24. OMG.org: Business Process Modeling Notation. 1.0 edn. (2006)
25. Aalst, W., Hee, K.: Workflow Management. MIT Press, Cambridge (2002)
26. Milner, R.: A Calculus of Communicating Systems. In: Jones, N.D. (ed.) Semantics-Directed Compiler Generation. LNCS, vol. 94, Springer, Heidelberg (1980)
27. Puhlmann, F.: A Tool Chain for Lazy Soundness. In: Demo Session of the 4th International Conference on Business Process Management, CEUR Workshop Proceedings, Vienna, vol. 203, pp. 9–16 (2006)
28. Basten, T.: In Terms of Nets: System Design with Petri Nets and Process Algebra. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (1998)

Extending BPMN for Modeling Complex Choreographies

Gero Decker and Frank Puhlmann

Business Process Technology Group
Hasso Plattner Institut for IT Systems Engineering
University of Potsdam
D-14482 Potsdam, Germany
{gero.decker,frank.puhlmann}@hpi.uni-potsdam.de

Abstract. Capturing the interaction behavior between two or more business parties has major importance in the context of business-to-business (B2B) process integration. The Business Process Modeling Notation (BPMN), being the de-facto standard for modeling intra-organizational processes, also includes capabilities for describing cross-organizational collaboration. However, as this paper will show, BPMN fails to capture advanced choreography scenarios. Therefore, this paper proposes extensions to broaden the applicability of BPMN. The proposal is validated using the Service Interaction Patterns.

1 Introduction

With the rise of service-oriented architectures (SOA [1]), business process definitions are more and more used as configuration artifacts for information systems. Services, being loosely coupled components described in a uniform way, ideally have such a granularity that they have business meaning. These services can be orchestrated in executable business processes, e.g. described in BPEL [2]. This enables an organization to quickly adapt to changing requirements and business environments. Especially in inter-organizational settings, interconnected business processes realized as services are at the center of attention. This calls for languages suited for describing the interaction behavior between the different services (a.k.a. the service choreography). Examples for such languages are Let's Dance [3] and WS-CDL [4]. Our aim is to use the popular Business Process Modeling Notation (BPMN [5]) as choreography language.

The Service Interaction Patterns [6] describe a set of recurrent choreography scenarios. They range from simple message exchanges to scenarios involving multiple participants and multiple message exchanges. These patterns can be used to evaluate choreography languages. Although the BPMN allows to define choreographies through a swimlane concept and a distinction between control flow and message flow, it only provides direct support for a limited set of the patterns. This papers discusses these deficiencies and increases the suitability of BPMN for choreography modeling by introducing language extensions.

The remainder of this paper is structured as follows. The next section will introduce a choreography example and assess BPMN for its pattern support. Section 3 gives an overview of the proposed extensions, before section 4 validates the extensions by investigating on their pattern support and section 5 further discusses our results. Section 6 reports on related work and finally section 7 concludes and gives an outlook to future work.

2 Assessment of BPMN

Figure 1 shows an auctioning scenario represented in BPMN. It involves three roles, namely seller, bidder and auctioning service. Every time a seller decides to initiate an auction, it sends an auction creation request to the auctioning service. This triggers the instantiation of the auctioning service's process. The auction is scheduled to start at a defined point in time. Once this moment is reached, different bids are received by the auctioning service. Bids are placed by different bidders and an individual bidder is allowed to place several bids. The latter allows a bidder to react on higher bids from other bidders. Once the auction is over, it is checked if at least one bid has been received. If this is not the case, the auctioning service informs the seller and the choreography instance (a.k.a. conversation) ends. Otherwise, the winning bidder is selected and the seller is informed about who won. Those bidders who were not successful are notified. The winning bidder is informed, which finally leads to payment and the delivery of the goods.

The figure illustrates the different participant behavior descriptions which are interconnected through message flow. We omitted the (required) BPMN end events due to space reasons. We represented the receipt of multiple bids via a loop marker in the "receive bid" task. The end of the auction is denoted with an intermediate timer event attached to the receive and send bid activities of the auctioning service and the bidder. We used the event-based gateway to route the sequence flows of the seller and the bidder according to the outcome of the auction. Furthermore, we used the multiple instances marker to represent the parallel emission of all sorry messages.

The resulting BPMN diagram captures the processes of each participant, the bidder, the auctioning service, as well as the seller. However, several aspects could not be captured.

- **Multiplicity of participants.** In our scenario, several bidders take part in one conversation. All bidders must conform to the same interaction behavior as depicted in the BPMN diagram. However, in addition to the mere fact that we have many bidders involved, we need to distinguish them: Only one bidder can win the auction. It is only her to receive the completion message, whereas the others receive sorry messages. It is only her to perform payment and to receive the product.
- **Correlation.** The auctioning service receives messages from different bidders. As we are dealing with an asynchronous setting, it is essential for a participant to correlate messages exchanged with the same interaction partner. Imagine more complex sub-conversations between the auctioning service

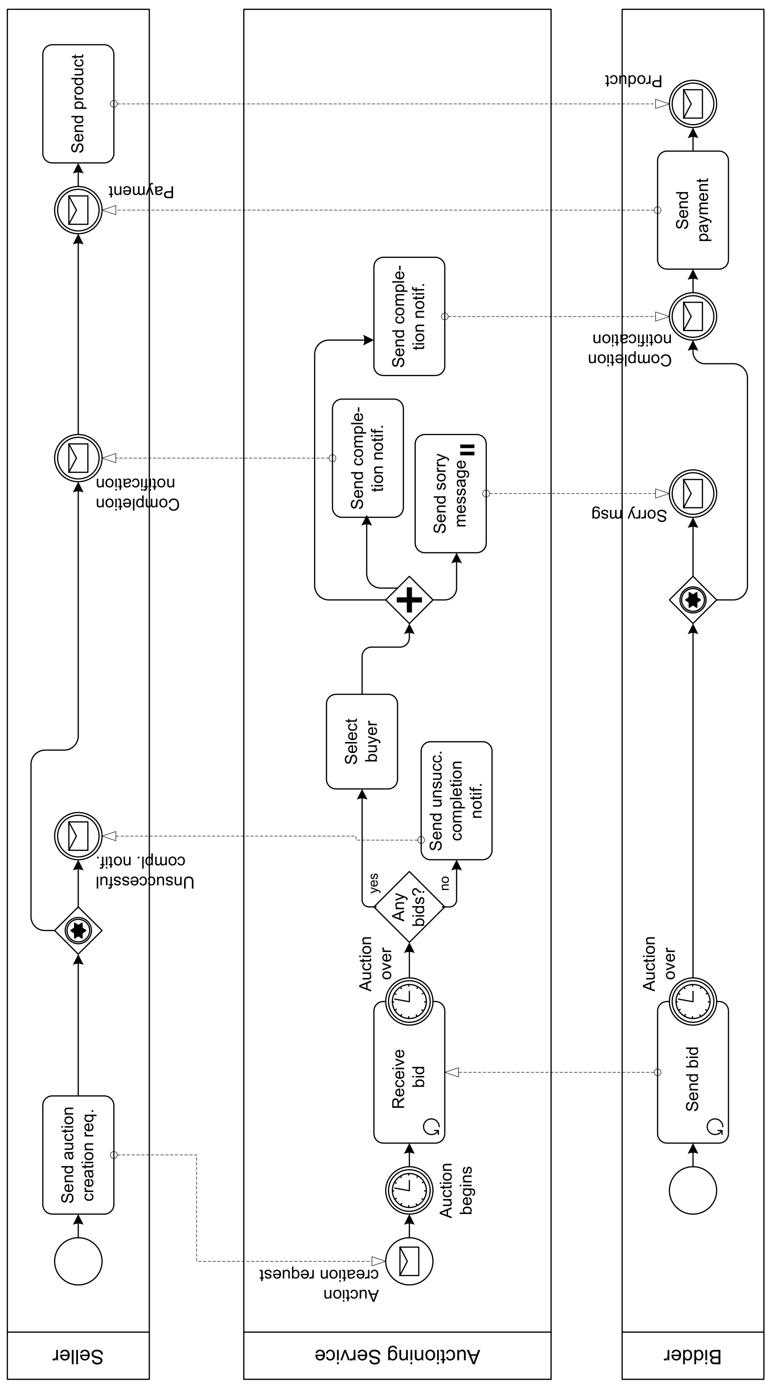


Fig. 1. BPMN choreography describing an auctioning scenario

and the bidders. Here, the different sub-conversations with the different bidders need to be distinguished from each other.
– **Participant reference passing.** The winning bidder—the buyer—needs to contact the seller that is former unknown to her. To make this happen, she somehow needs to gain knowledge about the seller. Hence, the auctioning service passes the reference to the seller to her. In turn, the seller needs to make sure that she only accepts payment from the winning bidder. This can only be ensured if the auctioning service actually tells her who has won.

None of the these requirements can be properly represented in BPMN. This has an effect on BPMN's support for the Service Interaction Patterns as these requirements also appear in the set of patterns.

Three of the four "single-transmission multilateral interaction patterns" involve a set of participants, where the exact number might only be known at runtime. Therefore, BPMN does not support this group of patterns (except for *Racing Incoming Messages*, which is directly supported through the event-based gateway). The multiplicity problem also applies to *Contingent Request*, where a participant sends a request to another participant. If this participant does not respond within a given timeframe, a third participant is contacted. As the length of the list of potential recipients of request might not be known at design-time, BPMN does not support this pattern.

Request with Referral involves participant reference passing. Also *Relayed Request* might involve reference passing. Here, a participant A makes a request to a participant B who delegates it to yet another participant C. C subsequently interacts with A, while B observes a view of the interactions. As C might not know A in advance, B might need to send the reference of A when delegating the request. Therefore, both *Request with Referral* and *Relayed Request* are not supported in BPMN. In analogy to [7], we do not consider *Dynamic Routing* in this assessment as the pattern description is too imprecise.

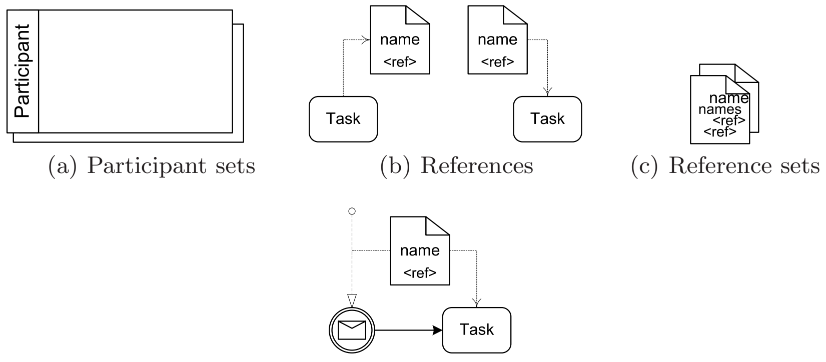
Only patterns *Send*, *Receive*, *Send/receive*, *Racing Incoming Messages* and *Multi-responses* are directly supported in BPMN. Therefore, we present BPMN extensions that overcome the illustrated issues in the next section.

3 BPMN Extensions

We introduce extension for the BPMN that allow the representation of multiple participants, correlations, and reference passing.

3.1 Participant Sets

Pools can represent individual participants in BPMN. As we have seen in the previous section we need to distinguish those cases where at most one participant of a particular participant type is involved in one conversation or if there can be potentially many participants involved. In our auctioning example, there is exactly one seller and one auctioning service involved in one conversation. However, we have potentially many bidders involved.



(a) Participant sets (b) References (c) Reference sets

(d) Reference passing

Fig. 2. BPMN extensions

For representing multiple participants we introduce shadowed pools as new notational element, shown in figure 2(a). A set of participants of the same type involved in the same conversation is called a *participant set*.

3.2 References

The main challenge with participant sets is that we need to distinguish individual participants out of this set. We do this via references as shown in figure 2(b). A reference is a special data object enhanced with ⟨ref⟩. A reference can be connected to a flow object via associations. We give the following semantics to the different connection directions:

– A reference can be written by a flow object (represented by an association from the flow object to the reference). (i) If the flow object is a receive activity, e.g. an intermediate message event or an activity with incoming message flow, the reference will point to the sender upon message receipt. If the reference already pointed to a participant, the reference will simply be overwritten. (ii) If the flow object is not a receive activity, it is not specified what participant the reference will point to. Consider the selection of the buyer in our example.
– A reference can be read by a flow object (represented by an association from the reference to the flow object). (i) If the flow object is a send activity, the message will be sent to the participant the reference points to. In our example the auctioning service sends a completion notification to exactly that bidder out of the bidder set, who was selected to have won the auction. (ii) If the flow object is a receive activity, then a message is only awaited from the defined participant. E.g. the seller only waits for payment from the buyer. (iii) If the flow object is neither a send nor a receive activity, it is not specified what happens with that reference inside the activity.

References cover those cases where an individual participant needs to be identified. However, we might need to select subsets of the participants involved in one conversation. In our example, this is the case for those bidders who did not win the auction. We need to send a sorry message to all of them—but we must not send this message to the winning bidder. We introduce reference sets as shown in figure 2(c) with the following semantics:

- A reference set can be modified by a flow object (represented through an association from the flow object to the reference set). (i) If the flow object is a receive activity, a reference to the sender of the message will be added to the reference set if such a reference is not already contained in the set. In our example we find a "receive bids" activity where bids from different bidders are received. In case a bidder who has already placed a bid in the same auction places another bid, no reference will be added to the set. However, if a new bidder takes part, a reference will be added. (ii) If the flow object is not a receive activity, it is not specified, what exactly happens with the set. It might be overwritten completely or references might be added, removed or changed.
- A reference set can be read by a flow object. (i) If the activity is a looped activity, i.e. a sequential loop or a multiple-instances activity, the reference set determines the number of repetitions or instances. This requires that at most one reference set serves as input for a looped activity. A special case is a looped send activity. Here, a message is sent to every of the referenced participants. In those cases, where the looped activity is a complex activity, a reference can be placed inside this activity which will represents the selected reference out of the set for a particular instance or repetition. (ii) If the activity is not a looped activity, it is not specified how the reference set is used within the activity. In our example, the "select buyer" activity takes the reference set as input and selects the winning bidder.

3.3 Reference Passing

References can be passed to other participants as shown in figure 2(d). The reference is connected to a message flow with an undirected association. The passed reference can be connected to other flow objects with directed associations. In figure 2(d), the passed reference is used in the task.

3.4 Example

The example from figure 1 is extended with the proposed extensions, shown in figure 3. First of all, a shadow was added to the pool of the bidder to represent a participant set. The "receive bid" task of the auctioning service collects the references of the different bidders into a reference set. The reference set is forwarded to the "select buyer" task. Inside this task, the successful bidder is selected and placed into a new reference, denoted as buyer. The remaining references of the bidders reference set are placed into an others reference set. The others reference set is used as an input to the "send sorry message" task. Here, an instance is created for each element of the set. Hence, all unsuccessful bidders are notified. The buyer reference is forwarded to the "send completion

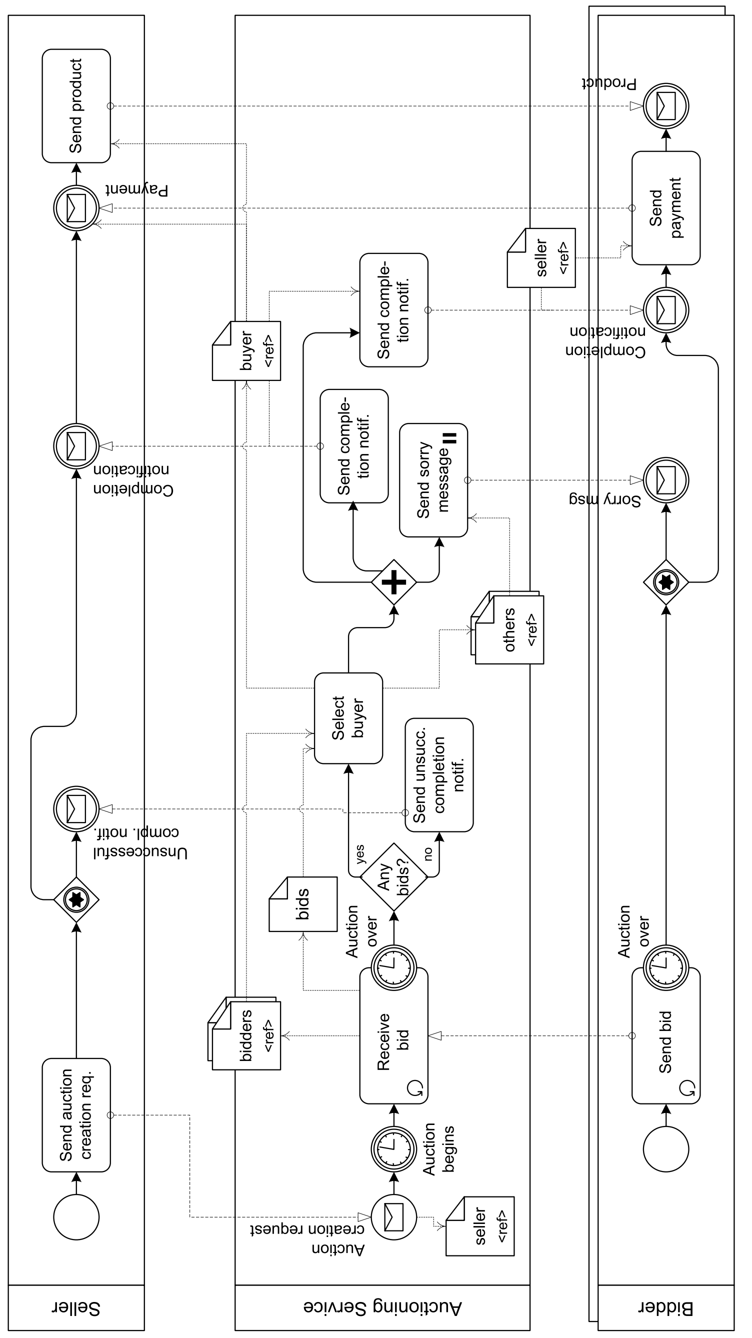


Fig. 3. The auctioning scenario represented using the extended BPMN

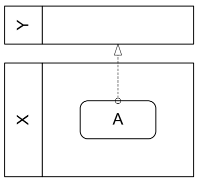
notification" task, where it determines the instance of the bidder that should be contacted. Furthermore, it is passed to the seller, where it is used as an input for the reception of the payment as well as determining the reference of the bidder's instance to which the product should be sent. Finally, a reference of the seller is passed to the successful bidder. This reference is acquired implicitly via the initial interaction between the seller and the auctioning service.

4 Validation

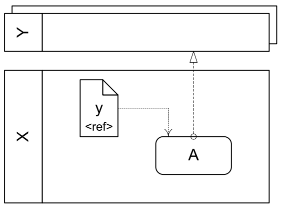
This section validates the proposed BPMN extensions by investigating how the Service Interaction Patterns can be represented. It is notable that many of the patterns require multiple participants and/or dynamic binding of interaction partners via reference passing.

4.1 Single Transmission Bilateral Interaction Patterns

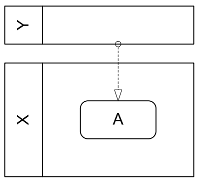
The single transmission bilateral interaction patterns represent basic interaction behavior. Graphical representations are shown in figure 4.



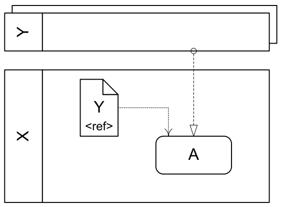
(a) Send



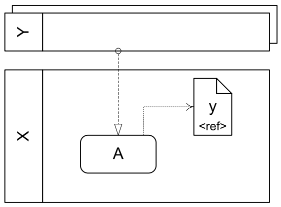
(b) Send to Reference



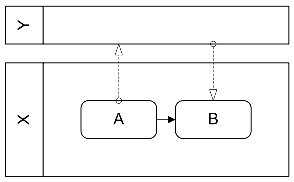
(c) Receive



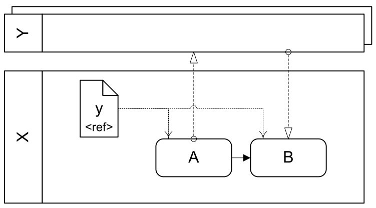
(d) Receive from Reference



(e) Receive Reference



(f) Send/Receive



(g) Send to / Receive from Reference

Fig. 4. Single transmission bilateral interaction patterns

Send: A process sends a message to another process. The *Send* interaction pattern is depicted in figure 4(a). It is an assumption that the receiver gains knowledge about the reference of the requester. If the message flow is targeted at a participant set, the matching instance has to be determined via a reference, shown in figure 4(b).

Receive: A process receives a message from another process. The *Receive* interaction pattern is depicted in figure 4(c). According to the previous pattern, the receiver automatically gains knowledge about the reference of the requester. If the message should be received from a particular instance of a participant set, a reference according to figure 4(d) has to be used. If a message is received from an unspecified instance of the participant set, the corresponding reference can be collected, shown in figure 4(e).

Send/Receive: A process X engages in two causally related interactions. In the first interaction X sends a message to another process Y (the request), while in the second one X receives a message from Y (the response). A combined send/receive interaction is shown in figure 4(f). Once again, due to a one to one multiplicity, the correlation between requester and provider is evident. If the interaction partner is a certain instance of a participant set, a reference according to figure 4(g) has to be used.

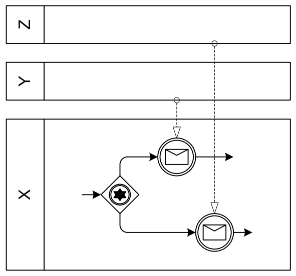
4.2 Single Transmission Multilateral Interaction Patterns

The single transmission multilateral interaction patterns represent one to many or many to one interactions. Graphical representations are shown in figure 5.

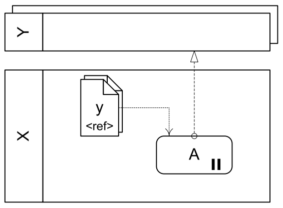
Racing Incoming Messages: A process expects to receive one among a set of messages. These messages may be structurally different (i.e. different types) and may come from different categories of processes. The way a message is processed depends on its type and/or the category of processes from which it comes. Figure 5(a) shows the solution to this pattern. If several instances of a participant set should be used instead of Y and Z, a single receive task is sufficient.

One-to-many Send: A process sends messages to several other processes. The messages all have the same type (although their contents may differ). This pattern is depicted in figure 5(b). The multiple instance task A sends a message to each reference contained in the reference set. We assume that all participants referenced are of the same type.

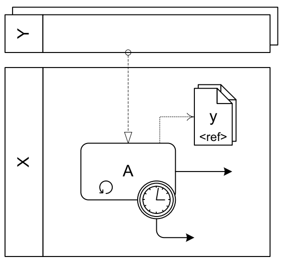
One-from-many Receive: A process receives a number of logically related messages that arise from autonomous events occurring at different processes. The arrival of messages needs to be timely so that they can be correlated as a single logical request. The one-from-many receive pattern is shown in figure 5(c). The references of the senders are collected in a reference set created in the loop-type task A. If enough messages have been gathered (decided internally inside A), the standard outgoing sequence flow is activated. If instead a timeout occurred, the interaction failed.



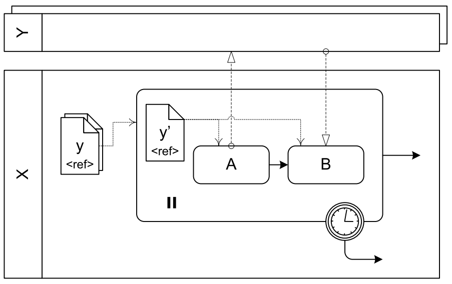
(a) Racing incoming messages



(b) One-to-many send



(c) One-from-many receive



(d) One-to-many send/receive

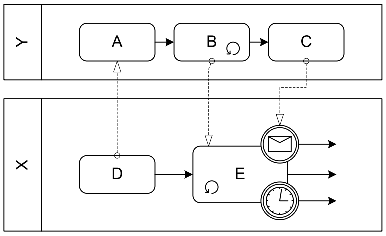
Fig. 5. Single transmission multilateral interaction patterns

On-to-many send/receive: A process sends a request to several other processes, which may all be identical or logical related. Responses are expected within a given timeframe. However, some responses may not arrive within the timeframe and some processes may even not respond at all. The *One-to-many Send/receive* pattern is shown in figure 5(d). The associated reference set points to the participants that should be included. Like in the preceding pattern, also in this pattern the task B decides if enough responses have been gathered in the given timeframe. The figure includes a reference *y'* used within the sub-process. This reference is to be filled for every instance that is spawned, as already mentioned in section 3.2.

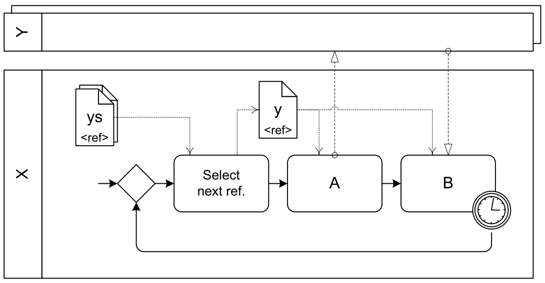
4.3 Multi Transmission Interaction Patterns

The multi transmission interaction patterns represent many to many interactions. Graphical representations are shown in figure 6.

Multi-responses: A process X sends a request to another process Y. Subsequently, X receives any number of responses from Y until no further responses are required. The trigger of no further responses can arise from a temporal condition or message content, and can arise from either X or Y's side. This pattern is depicted in figure 6(a). The task D of X sends an initial request to task A of Y.



(a) Multi-responses



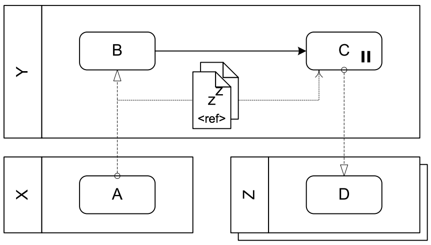
(b) Contingent requests

Fig. 6. Multi transmission interaction patterns

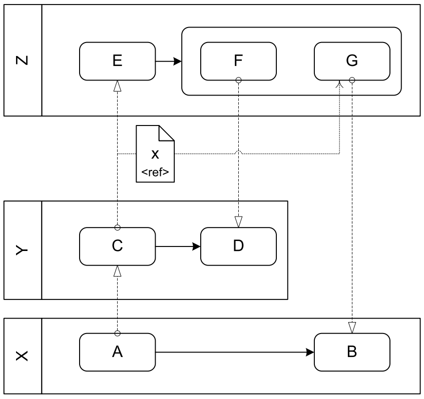
Task B of Y responds until they are no more responses. Task E in X receives the responses until (1) a timeout occurs, (2) E decides to have gathered enough responses, or (3) a stop messages arrives from Y.

Contingent Requests: A process X makes a request to another party Y. If X does not receive a response within a certain timeframe, X alternatively sends a request to another process Z, and so on. This pattern is shown in figure 6(b). Initially, a reference set is passed to a task that selects a certain reference out of the set. The downstream task A receives this reference and initiates a request. Task B tries to receive the response. If no response is received in the given timeframe, another reference out of the reference set is selected and processed as described. What cannot be captured with our extensions, however, is the reception of messages from previous requests that failed due to a timeout.

Atomic Multicast Notification: A process sends notifications to several parties such that a certain number of parties are required to accept the notification within a certain timeframe. This pattern requires transactional behavior spanning multiple processes. Transactions are included in BPMN, however, they must only be applied within one process. Distributed transactions are not supported. Therefore, we can only provide a workaround for this pattern in our extended BPMN. It looks similar to *One-to-many Send/receive* with a completion condition at the notifying side.



(a) Request with Referral



(b) Relayed Request

Fig. 7. Routing patterns

4.4 Routing Patterns

The routing patterns describe flexible interaction behavior between a set of processes. Graphical representations are shown in figure 7.

Request with Referral: Process X sends a request to process Y indicating that any follow-up response should be sent to a number of other processes (Z1, Z2, . . . , Zn) depending on the evaluation of certain conditions. The solution to this pattern is shown in figure 7(a). It uses reference passing to denote the instances of Z that should receive the follow-up responses.

Relayed Request: Process X makes a request to process Y which delegates the request to other processes (Z1, . . . , Zn). Processes Z1, . . . , Zn then continue

Table 1. BPMN vs. extended BPMN

Pattern	**BPMN**	**ext. BPMN**
Send	+	+
Receive	+	+
Send/Receive	+	+
Racing Incoming Messages	+	+
One-to-many Send	-	+
One-from-many Receive	-	+
One-to-many Send/Receive	-	+
Multi-reponses	+	+
Contingent Request	-	+/-
Atomic Multicast Notification	-	-
Request with a Referral	-	+
Relayed Request	-	+

interacting with process X while process Y observes a "view" of the interactions including faults. The interacting parties are aware of this "view". The *Relayed Request* pattern is shown in figure 7(b). While participant Z has immediate knowledge of Y, it needs a reference to participant X. This is received via reference passing from Y.

4.5 Validation Summary

A comparison on the supported Service Interaction Patterns for the standard BPMN as well as our proposed extension is shown in table 4.4. As already argued previously, we do not support *Atomic Multicast Notification* and did not consider *Dynamic Routing* in this assessment. *Contingent Requests* is also only partly supported, since (late) responses from earlier requests are ignored.

5 Discussion

Our proposals make heavy use of refined data objects. A major problem with BPMN data objects is that their semantics is not clearly defined in the BPMN specification. E.g. it is unclear what it means if different activities write on the same data object. Here, we simply assume that if an activity has write-access to a data object, it (might) overwrite the entire content of the data object upon completion. BPMN does not have the notion of collections or buffers, as they are present in UML Activity Diagrams [8]. Therefore, we introduced a distinction between simple data objects and data object sets, where we assume that write-access to a data object set typically means that the activity (might) add an object to the set. We do not require that data objects are only placed within pools or only accessed from within one pool. However, we have to leave a detailed discussion on BPMN data objects and their semantics to future work.

The BPMN extensions presented in this paper are aligned with the work done on BPEL4Chor [9], an extension to abstract BPEL for modeling choreographies. This becomes evident in the semantics of references and reference sets. Awaiting messages from any sender vs. awaiting messages from a particular sender expressed through the absence or presence of a read-relationship between references and receive activities is analogous to the semantics of BPEL4Chor participant references that are either uninitialized or already set. Furthermore, the notion of adding references to a reference set in case a message is received from a sender that is not yet referenced in the set, is analogous to the notion of containment of a BPEL4Chor participant reference in a participant reference set. However, a detailed transformation of our extended BPMN to BPEL4Chor goes beyond the scope of this paper and must be left to future work.

References express correlation in those cases where receive activities read references. This defines who messages are to be received from. However, this notion of correlation only covers a limited set of scenarios. Imagine settings, where the same pair of participants engage in different parallel conversations. Here, our notion of references is not sufficient to distinguish the different conversations.

Furthermore, it might be important to specify what message parts correlation is actually based on. E.g. a customer id or a shipment invoice number might be used as concrete correlation identifiers. There might be even more sophisticated correlation mechanisms needed, such as ranges of values or time-based correlation of messages. [10] provides a set of correlation patterns that might be a starting point for further refinements for correlation support.

In this paper we have left BPMN unchanged as much as possible while providing increased support for the Service Interaction Patterns. However, there is a general discussion whether the *interconnection modeling* approach, as it is the case for BPMN, is suited for choreography modeling at all. We have seen that we basically define control flow on a per-participant basis. Corresponding send and receive activities are connected through a message flow, jointly representing interactions.

An alternative to this approach is *interaction modeling*, where atomic interactions are the basic building blocks and control and data flow is defined between them. The main advantage of this approach is that incompatibility between different participants cannot occur in choreography models. It also reduces the number of modeling elements for representing a certain choreography. This increases modeling speed and helps to keep the models readable. Control and data flow dependencies are defined from a global perspective in the sense that (for most constructs) it does not need to be expressed explicitly, to what particular participant it actually belongs. Techniques for generating participant behavior descriptions out of the interaction model then care about which participant actually has to enforce a certain dependency later on.

Sometimes it is not possible to generate participant behavior descriptions such that all dependencies in the choreography are collectively enforced by them without adding synchronization interactions. Such choreographies are called *locally unenforceable*. For details please refer to [11] and [12]. A detailed comparison between interconnection models and interaction models goes beyond the scope of this paper and needs to be discussed in future work.

6 Related Work

BPMN enjoys widespread use in both industry and academia. [13] delivers an assessment of BPMN regarding its support for the Workflow Patterns [14] as well as its capabilities for the data and resource allocation perspective. However, this assessment does not include the Service Interaction Patterns.

A range of languages where introduced for modeling choreographies. BPEL4-Chor [9] adds a thin layer on top of abstract BPEL, interconnecting the different participant behavior descriptions. Let's Dance [3] and WS-CDL [4] follow the interaction modeling approach as described in the previous section. Like BPMN, Let's Dance is mainly targeted at business analysts and comes with a graphical notation. WS-CDL is tightly linked to other web services standards such as WSDL. Both languages have been assessed for their Service Interaction Pattern support (cf. [7]). It turns out that Let's Dance directly supports most patterns.

WS-CDL is a little less suited for choreography modeling as it only comes with limited support for expressing those scenarios where multiple participants of the same type are involved in a conversation and the actual number of participants is only known at runtime. Event-driven Process Chains (EPC) is another widely-used process modeling language. In [15] extensions for inter-organizational process modeling are proposed. However, there has not been an assessment using the Service Interaction Patterns regarding their suitability.

There has also been work on mapping (subsets of) BPMN to formalisms. Dijkman et al. present a mapping to Petri nets in [16], enabling the verification of soundness and liveless. However, the formalization does not include message flows. Therefore, reasoning on choreographies is out of scope of their work. Wong et al. present another formalization of BPMN based on Communicating Sequential Processes (CSP) in [17]. In [18] they then show how compatibility checking can be carried out for BPMN choreographies. Other approaches for compatibility checking in choreographies are introduced by Martens [19], Puhlmann et al. [20] and Massuthe et al. [21]. A general introduction into the different viewpoints of choreographies can be found in [22].

7 Conclusion

In this paper we have identified weaknesses of BPMN regarding its suitability for choreography modeling. We based our assessment on the Service Interaction Patterns and concluded that there is direct support for only five out of the twelve patterns considered. We then proposed extensions to overcome these limitations and validated the extended BPMN with the patterns.

In future work we are going to introduce a formal mapping for the new concepts. This enables the verification of complex choreographies, including compatibility and conformance checking. In [23] we have already shown that name creation and restriction in π-calculus are useful concepts for formalizing choreographies. Therefore, we consider using π-calculus or a Petri net version enhanced with a name concept, e.g. similar to ν-nets as presented in [24], as formal basis. The latter would enable us to reuse and extend the Petri-nets-mapping in [16].

References

1. Burbeck, S.: The Tao of E-Business Services (2000)
2. Fallside, D.C., Walmsley, P.: Web Services Business Process Execution Language Version 2.0. Technical report (2005), http://www.oasis-open.org/apps/org/workgroup/wsbpel/
3. Zaha, J.M., Barros, A., Dumas, M., ter Hofstede, A.: A Language for Service Behavior Modeling. In: CoopIS 2006. Proceedings 14th International Conference on Cooperative Information Systems, Montpellier, France, Springer, Heidelberg (2006)

4. Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation. Technical report (2005), http://www.w3.org/TR/ws-cdl-10
5. OMG.org: Business Process Modeling Notation. 1.0 edn. (2006)
6. Barros, A., Dumas, M., Hofstede, A.: Service Interaction Patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 302–318. Springer, Heidelberg (2005)
7. Decker, G., Overdick, H., Zaha, J.M.: On the Suitability of WS-CDL for Choreography Modeling. In: EMISA 2006. Proceedings of Methoden, Konzepte und Technologien für die Entwicklung von dienstebasierten Informationssystemen, Hamburg, Germany (2006)
8. Object Management Group (OMG): UML 2.0 Superstructure Specification (2005)
9. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4chor: Extending BPEL for Modeling Choreographies. In: Proceedings International Conference on Web Services (ICWS) (2007)
10. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation Patterns in Service-Oriented Architectures. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, Springer, Heidelberg (2007)
11. Zaha, J.M., Dumas, M., ter Hofstede, A., Barros, A., Decker, G.: Service Interaction Modeling: Bridging Global and Local Views. In: EDOC 2006. Proceedings 10th IEEE International EDOC Conference, Hong Kong, IEEE Computer Society Press, Los Alamitos (2006)
12. Decker, G., Weske, M.: Local Enforceability in Interaction Petri Nets. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, Springer, Heidelberg (2007)
13. Wohed, P., van der Aalst, W.M., Dumas, M., ter Hofstede, A., Russell, N.: On the Suitability of BPMN for Business Process Modelling. In: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, Springer, Heidelberg (2006)
14. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
15. Seel, C., Vanderhaeghen, D.: Meta-model based extensions of the epc for inter-organisational process modelling. In: Proceedings 4th GI-Workshop EPK 2005 - Geschäftsprozessmanagement (2005)
16. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and automated analysis of BPMN process models. Preprint 7115, Queensland University of Technology, Brisbane, Australia (2007)
17. Wong, P.Y., Gibbons, J.: A process semantics for BPMN. Technical report, Oxford University Computing Laboratory (2007), http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmnsem.pdf
18. Wong, P.Y., Gibbons, J.: Verifying business process compatibility. In: MTCoord 2007. Proceedings 3rd International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems, Paphos, Cyprus (2007)
19. Martens, A.: Analyzing Web Service based Business Processes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, Springer, Heidelberg (2005)
20. Puhlmann, F., Weske, M.: Interaction Soundness for Service Orchestrations. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 302–313. Springer, Heidelberg (2006)
21. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1, 35–43 (2005)

22. Dijkman, R., Dumas, M.: Service-oriented Design: A Multi-viewpoint Approach. International Journal of Cooperative Information Systems 13, 337–368 (2004)
23. Decker, G., Puhlmann, F., Weske, M.: Formalizing Service Interactions. In: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, pp. 414–419. Springer, Heidelberg (2006)
24. Rosa-Velardo, F., de Frutos-Escrig, D.: Name creation vs. replication in petri net systems. In: Petri Nets 2007. Proceedings 28th International Conference on Application and Theory of Petri Nets and other Models of Concurrency, Siedlce, Poland (2007)

Semantics of Standard Process Models with OR-Joins

Marlon Dumas[1,2], Alexander Grosskopf[3], Thomas Hettel[4,1], and Moe Wynn[1]

[1] Queensland University of Technology, Australia
{m.dumas, m.wynn}@qut.edu.au
[2] University of Tartu, Estonia
marlon.dumas@ut.ee
[3] Hasso-Plattner Institute, Potsdam, Germany
alexander.grosskopf@hpi.uni-potsdam.de
[4] SAP Research Centre Brisbane, Australia
t.hettel@sap.com

Abstract. The Business Process Modeling Notation (BPMN) is an emerging standard for capturing business processes. Like its predecessors, BPMN lacks a formal semantics and many of its features are subject to interpretation. One construct of BPMN that has an ambiguous semantics is the OR-join. Several formal semantics of this construct have been proposed for similar languages such as EPCs and YAWL. However, these existing semantics are computationally expensive. This paper formulates a semantics of the OR-join in BPMN for which enablement of an OR-join in a process model can be evaluated in quadratic time in terms of the total number of elements in the model. This complexity can be reduced down to linear-time after materializing a quadratic-sized data structure at design-time. The paper also shows how to efficiently detect the enablement of an OR-join incrementally as the execution of a process instance unfolds.

1 Introduction

Business process management as a discipline has traditionally suffered from a proliferation of process definition languages based on similar but subtly different concepts and constructs. After numerous attempts, standardization efforts in this space have converged towards two languages: the Business Process Modeling Notation (BPMN) [13], which is intended for modeling business processes primarily during the analysis and design phases, and the Business Process Execution Language (BPEL) [9], which is intended for implementation and execution of business processes in a service-oriented architecture. The standard specifications of both of these languages are given in a narrative, informal style. In the case of BPEL, a number of formalizations have been proposed [3]. On the other hand, virtually no attempt has been made to attach a formal semantics to BPMN, barring recent work on formalizing subsets thereof [17,7]. Compounded with the fact that executability has not been a major concern during the standardization of BPMN, this has led to a standard specification with

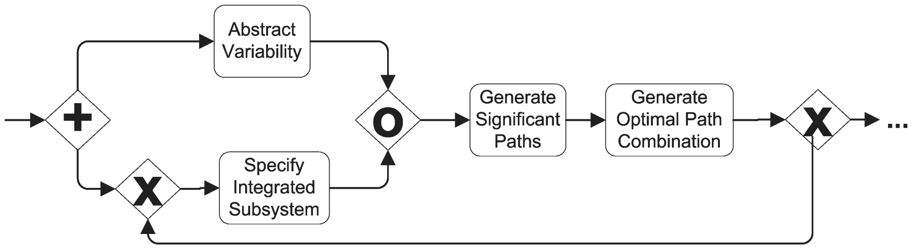

Abstract Variability
Specify Integrated Subsystem
Generate Significant Paths
Generate Optimal Path Combination
...

Fig. 1. Process fragment with an OR-join (example inspired from [14])

numerous ambiguities. This situation raises the risk that different tools adopt different interpretations of BPMN, especially those tools supporting process simulation and automated generation of executable process definitions which start to emerge [16].

In separate work, we formalized a subset of BPMN [7]. One of the constructs left aside in this formalization, as well as in reference [17], is the *inclusive merge gateway* (hereafter called the *OR-join*). This construct corresponds to a workflow pattern called "Synchronizing Merge" [2]. An OR-join is a point in a process where several branches converge. For each of its incoming branch, the OR-join will normally wait for a token indicating its completion; but if at some point in time it can be determined that no token will ever arrive along a given incoming branch, the OR-join will not wait for a token along that branch.

Figure 1 shows a use case of the OR-join. The first time this process fragment is executed, both tasks "Abstract Variability" and "Specify Integrated Subsystem" are executed in parallel. This parallel execution is captured by the AND-split gateway (a lozenge with a "+" symbol). The OR-join (lozenge with an "O" symbol) will then wait for both tasks to complete. Thereafter, the execution proceeds along task "Generate Significant Paths" followed by "Generate Optimal Path Combination". After completion of this latter task, a choice is made between repeating task "Specify Integrated Subsystem" or proceeding with the rest of the process (not shown in the figure). When the second execution of "Specify Integrated Subsystem" completes, the OR-join receives a token along one of its incoming branches. The OR-join can fire at this point without waiting for a token from task "Abstract Variability", because this task is not executed in the second round. In other words, the branch coming from task "Abstract Variability" into the OR-join is not "active", and thus the OR-join will not wait for a token along this branch.

Formalizing the OR-join is challenging as highlighted by previous experiences in defining semantics for languages that incorporate this construct, such as EPCs [10] and YAWL [19]. Unlike other routing constructs, the OR-join has a non-local semantics: in order to determine whether or not an OR-join is enabled, it is not sufficient to examine the presence of tokens in its immediate vicinity. Instead, enablement of an OR-join may depend on the presence or absence of

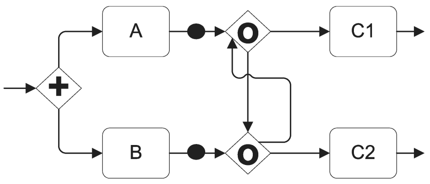


Fig. 2. Example of a vicious circle. Tokens are represented as black dots.

tokens in places far away in the model. Not surprisingly, the evaluation of enablement for OR-joins using existing semantics is computationally expensive.

Another issue when defining a semantics for the OR-join, is that the state of enablement of an OR-join (i.e. its ability to fire at a given point during the execution) may depend on the state of enablement of another OR-join in the model and vice-versa. In other words, two OR-joins may end up waiting for one another, a situation known as a *vicious circle* [1]. An example of a vicious circle is given in Figure 2. One may argue that such scenarios are hypothetical and there is no harm in excluding them or giving them an arbitrary semantics that generates deadlocks. However, in this paper we show that, without adding to the complexity, we can define a semantics for the OR-join in BPMN that is able to deal with such scenarios without generating deadlocks.

The contribution of this paper is a semantics of the OR-join in BPMN that strikes a balance between precision in determining when an OR-join should fire, and the computational complexity of determining whether or not an OR-join is enabled. After formulating this semantics, the paper presents an algorithm that allows enablement of a given OR-join in a model to be determined in quadratic time in terms of the total number of elements in the model. This complexity can be reduced to linear time if a quadratic-sized data structure is materialized at design-time for each OR-join in the model. Furthermore, we show that the proposed algorithm can be adapted to an incremental evaluation mode.

The rest of the paper is structured as follows. Some background on BPMN is given in Section 2. Next, the semantics of BPMN models with OR-joins is presented in Section 3, while Section 4 describes an algorithm for evaluating enablement of an OR-join in a given state. Section 5 introduces an incremental version of the algorithm that provides some additional optimization. Finally, Section 6 discusses related work while Section 7 concludes.

2 Syntax of BPMN

This section introduces the BPMN notation and provides an abstract syntax for BPMN process models.

2.1 BPMN Overview

A process model in BPMN is represented as a *Business Process Diagram* (BPD). A BPD is a graph in which nodes correspond to activities or routing steps

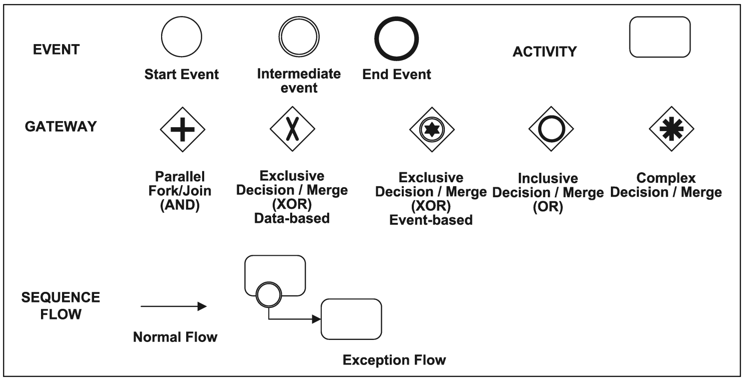


Fig. 3. BPMN graphical elements

(collectively called *objects*), while edges correspond to flows of control or flows of messages. Objects and flows can be grouped into pools and swimlanes to capture domains of responsibility. They can also be associated with artifacts that capture non-functional information. In this paper, we concentrate on the control-flow semantics of BPMN and thus we leave aside pools, swimlanes and artifacts. Also, since we only consider the semantics of one process model at a time, as opposed to the semantics of multiple communicating processes, we leave aside message flows. Figure 3 summarizes the constructs we focus upon.

An object can be an *event*, an *activity* or a *gateway*. An event, depicted as a circle, represents something that can affect a process execution. Events are classified based on their position in the graph into start events (events that are source in the graph), end events (events that are sink in the graph) and intermediate events. Events are also classified based on their triggering cause into timer events, message events, etc. Given that the cause of an event is not relevant from a control-flow perspective, we do not consider this latter classification.

An *activity*, depicted as a rounded rectangle, represents a unit of work. An activity may correspond to an undecomposed task or to a subprocess invocation. In this paper, we capture the execution of one individual process at a time. If this process invokes another one, the invoked subprocess is seen as a black-box, and accordingly, we treat its execution as that of an undecomposed task.

A *gateway* is used to control branching, forking, merging, and joining of paths and is represented using a diamond. There are different gateway types: (i) *exclusive gateways* for selecting one branch among a set of alternative branches based on data or events (XOR-split), or for merging a number of alternative branches (XOR-join); (ii) *parallel gateways* for forking one branch into multiple concurrent branches (AND-split) or for merging multiple concurrent branches into one (AND-join) using a barrier-synchronization policy; (iii) *inclusive gateways* for choosing one or multiple branches based on boolean expressions (OR-split), or for synchronizing multiple branches while waiting only for those branches that are active (OR-join); and (iv) *complex gateway* for modeling complex branching conditions and synchronization policies (e.g., wait for 3 branches out of 5).

Objects in a BPD are related by means of *sequence flows* and *exception flows*. A sequence flow captures a sequential dependency: when the task completes normally, a token is placed on each of its outgoing sequence flows. Meanwhile, if an error occurs during the performance of an activity, the activity is interrupted and a token is placed in the exception flow corresponding to that error.

2.2 Abstract Syntax

Abstracting from its concrete syntax, a BPD can be thought of as containing various types of objects and flows as captured by the following definition.

Definition 1 (Business Process Diagram (BPD)). *A BPD is a tuple* $\mathcal{BPD}$=($\mathcal{O}$, $\mathcal{A}$, $\mathcal{G}$, $\mathcal{E}$, $\mathcal{G}^{\mathcal{X}}$, $\mathcal{G}^{\mathcal{P}}$, $\mathcal{G}^{\mathcal{I}}$, $\mathcal{G}^{\mathcal{C}}$, $\mathcal{G}^{\mathcal{E}}$, $\mathcal{E}^{\mathcal{S}}$, $\mathcal{E}^{\mathcal{I}}$, $\mathcal{E}^{\mathcal{E}}$, $\mathcal{F}$, *abf*), *where*

- $\mathcal{O}$ *is a set of objects which can be partitioned into disjoint sets of activities* $\mathcal{A}$, *gateways* $\mathcal{G}$ *and events* $\mathcal{E}$,
- $\mathcal{G}$ *is a set of gateways which can be partitioned into disjoint sets of exclusive OR gateways* $\mathcal{G}^{\mathcal{X}}$, *parallel AND gateways* $\mathcal{G}^{\mathcal{P}}$, *inclusive OR gateways* $\mathcal{G}^{\mathcal{I}}$, *complex gateways* $\mathcal{G}^{\mathcal{C}}$ *and event-based gateways* $\mathcal{G}^{\mathcal{E}}$,
- $\mathcal{E}$ *is a set of events which can be partitioned into disjoint sets of start events* $\mathcal{E}^{\mathcal{S}}$, *intermediate events* $\mathcal{E}^{\mathcal{I}}$ *and end events* $\mathcal{E}^{\mathcal{E}}$,
- $\mathcal{F} \subseteq \mathcal{O} \times \mathcal{O}$ *is the flow relation between objects.*
- $abf : \mathcal{G}^{\mathcal{C}} \to \mathcal{O}$ *is a function capturing the preconditions for a complex gateway to be enabled, as discussed later in Section 3.2.*

The relation $\mathcal{F}$ defines a directed graph over the set of objects $\mathcal{O}$. For any object $x \in \mathcal{O}$, the set of direct predecessors is given by $pred(x) = \{y \in \mathcal{O} | y\mathcal{F}x\}$ and analogously the set of direct successors is given by $succ(x) = \{y \in \mathcal{O} | x\mathcal{F}y\}$. $\mathcal{F}^*$ is the reflexive transitive closure of $\mathcal{F}$. The set of all direct and transitive predecessors of an object x is given by function $pred^*(x) = \{y \in \mathcal{O} | y\mathcal{F}^*x\}$.

A BPD as defined in Definition 1 has no structural requirements in terms of a starting point or an end point. Typically, a business process model has one starting point, one or more end points and all the objects used in the modeled are connected. Definition 2 defines minimal structural requirements for a (connected) BPD, i.e., there is a start event, there is one or more end events and every object is on a path from the start event to an end event.

Definition 2 (Connected BPD). *A BPD* = ($\mathcal{O}$, $\mathcal{A}$, ...) *is connected if it satisfies the following conditions:*

- *there is exactly one start event with zero incoming flow and at least one outgoing flow:* $|\mathcal{E}^{\mathcal{S}}| = 1 \wedge \exists s \in \mathcal{E}^{\mathcal{S}} pred(s) = \emptyset \wedge |succ(s)| \geq 1$
- *there is one or more end events with incoming flows and zero outgoing flow:* $|\mathcal{E}^{\mathcal{E}}| \geq 1 \wedge \forall e \in \mathcal{E}^{\mathcal{E}} \; |pred(e)| \geq 1 \wedge succ(e) = \emptyset$
- *every object (other than the start and end events) is on a path from a start event to an end event:* $\forall x \in \mathcal{O} \setminus (\mathcal{E}^{\mathcal{S}} \cup \mathcal{E}^{\mathcal{E}}) \; \exists s \in \mathcal{E}^{\mathcal{S}} \; \exists e \in \mathcal{E}^{\mathcal{E}} \; s\mathcal{F}^*x \wedge x\mathcal{F}^*e$.

We assume that all BPDs are connected. We also assume, without loss of generality, that every gateway in a BPD is either a split gateway (it has one incoming flow and multiple outgoing flows) or a join gateway (multiple incoming flows and one outgoing flow). Similarly, we assume that activities and intermediate events have only one incoming flow and one outgoing flow. It is straightforward to expand a BPD that does not satisfy these conditions into one that does by applying simple expansion rules. For example, if an activity has multiple incoming flows it is equivalent to a structure where these flows lead to an XOR-join gateway and this gateway has a flow that leads into the activity in question. The above conditions together with the connectedness requirement entail that no activity or event in a BPD is both the source and target of the same flow.

We have excluded exception flows from the abstract syntax. To simplify the presentation of the proposed OR-join semantics and without loss of generality, we assume that a BPD is pre-processed as follows in order to replace all exception flows with gateways and sequence flows. If an activity contains at least one exception flow, all the outgoing flows of this activity are deleted and replaced by one single sequence flow that leads to an exclusive decision gateway (i.e. an XOR-split). This XOR-split has multiple branches: one branch corresponds to the sequence flow going out of the activity, and the other branches correspond to the various exception flows, and these exception flows are replaced with normal sequence flows. The idea is that once an activity completes (whether normally or abnormally) a decision is made to determine if the sequence flow is taken (if the task completed normally), or one of the exception flows is taken (if the task completed due to an error). This decision is captured by such a decision gateway, and in doing so, all the exception flows are replaced with sequence flows. Hereafter, we will use the term *flow* to refer to a *sequence flow*.

3 Semantics of BPMN Models

This section formulates a semantics for the OR-join in BPMN. After some considerations regarding the notion of the OR-join, the section introduces a formal definition of enablement of objects in BPMN process models with OR-joins.

3.1 Informal Definition

The BPMN specification makes vague statements about the meaning of the OR-join, such as: "the Inclusive Gateway [...] will wait for (synchronize) all Tokens that have been produced upstream. It does not require that all incoming Sequence Flow produce a Token (as the Parallel Gateway does). It requires that all [Tokens] that were actually produced by an upstream [object arrive]."[1] The clarity of this statement is hampered by the fact that the term "upstream" is not defined, and its meaning is unclear if there are cycles in the graph. Also, this statement does not clarify how many tokens from each incoming flow should the OR-join wait for.

[1] This sentence is incomplete in the specification, so we have added the last two words.

From a detailed reading of the BPMN specification, and from the definition of the Synchronizing Merge pattern [2] to which the BPMN specification refers to, we can distill the following characteristics of the OR-join:

– In line with the definition of all other gateways in BPMN, a necessary condition for an OR-join to be enabled is that there is at least one token in at least one of its incoming flows.
– A sufficient condition for an OR-join to be enabled is that there is at least one token in each of its incoming flows, i.e. all branches have "completed".
– If an incoming flow of the OR-join has no token, a necessary condition for the OR-join to be enabled is that it is not possible for a token to reach this flow. This captures the notion of waiting for tokens produced "upstream".

Thus an OR-join has a behavior in-between the XOR-join and the AND-join. The XOR-join waits for one token in one of its incoming flows, while the AND-join waits for one token in each of its incoming flows. The OR-join may behave like an XOR-join, or like an AND-join, or like something in-between depending on the state of the process instance. Another characteristic of the OR-join is that it only waits for tokens that will eventually arrive. As a result, an OR-join will not deadlock in situations where an AND-join would. For example, if we replaced the OR-join in Figure 1 with an AND-join, a deadlock would occur if task "Specify Integrated Subsystem" was repeated.

We decompose the control-flow semantics of objects in BPMN into an *enablement rule* and a *firing rule*. The enablement rule determines if the object is ready to fire in a given state. If one or more objects are enabled, the execution environment may select any one of them and fire it. The firing rule determines what happens then. For example, when an AND-join fires, it consumes one token from each of its incoming flows and it produces one token in its outgoing flow. On the other hand, when an OR-join fires, it consumes one token from each incoming flow that has a token, and it produces a token in the outgoing flow. The definition of firing rules for the various types of BPMN objects, including gateways, does not pose major challenges. Accordingly, we focus on the enablement rules, especially the one for the OR-join.

Given the above characteristic of the OR-join, we propose the following OR-join enablement rule:

> An OR-join o is enabled if there is at least one token in one of its incoming flows, and for each of its incoming flows f, either f has at least one token or, taking as an assumption that o will not fire, no token will arrive to flow f through a sequence of firings starting from the current state.

One variable in this informal definition is how to determine that no token will ever arrive to flow f. In the semantics of the OR-join for YAWL [18], a far-sighted approach is taken. Conceptually, the entire set of possible future states is computed to determine if, in any of these possible states, there will be at least one token in the incoming flow in question. This far-sighted approach ensures that the OR-join is enabled as early as possible. But its computational complexity is proportional to the number of possible states, as opposed to the number of elements in the model. To avoid this computational problem, we propose a "myopic" (or "short-sighted") approach: we can determine that no token will ever arrive to flow f, if we can determine that none of the direct or indirect predecessors of f is enabled.

This definition also needs some refinement to deal with cycles containing OR-joins. Hereafter, we say that two OR-joins are in *structural conflict* if one is a predecessor of the other and vice-versa. At a given state, two OR-joins are said to be in *partial conflict* if they are in structural conflict and each of them has at least one token in at least one of its incoming flows. Finally, two OR-joins are said to be in *full conflict* iff they are in structural conflict and they are only waiting for each other to fire first so that they can become enabled. This scenario was illustrated in Figure 2. Also, if an OR-join is part of a loop, this OR-join is in structural conflict with itself, and if some of its input flows contain a token, the OR-join may end up being in full conflict with itself.

With respect to the above enablement rule, partial and full conflicts raise the following issue: In the process of determining whether the OR-join o_1 is enabled, we may need to recursively ask ourselves the question of whether o_1 is enabled; or in the process of determining whether o_1 is enabled, we ask the question of whether another OR-join o'_1 is enabled, and in determining whether o'_1 is enabled we ask the question of whether o_1 is enabled. At least two approaches are possible to break such vicious circles [19]. We could be "optimistic" and say that o_1 is enabled if it has at least one of its input flow marked. Or we can be "pessimistic" and say that o_1 is enabled if and only if tokens are present at each of its incoming branches. Adopting an optimistic strategy can lead to deadlocks [20], for example in the full conflict depicted in Figure 2. Accordingly, in this paper, we adopt a pessimistic strategy: in the case of a partial or full conflict between o_1 and o'_1, gateway o_1 will treat o'_1 as not enabled, and reciprocally, o'_1 will treat o_1 as not enabled. Consequently, both o_1 and o'_1 will be enabled if they are in full conflict.

3.2 Formal Definition of Enablement

From a control-flow perspective, the state of an instance of a BPD can be captured as a set of tokens distributed among the flows composing the BPD. We can think of a flow $f \in \mathcal{F}$ as a place where tokens are stored. Initially, one token is located in each of the flows emanating from the start event of a BPD. As the execution of the BPD proceeds, tokens are removed from certain flows and added to other flows according to the firing rules. Hence, the state of a process instance at a given state can be captured using a token count function $tc : \mathcal{F} \rightarrow \mathbb{N}$ that takes as input a flow and returns the number of tokens stored in that flow. Flows are represented as pairs of objects (o_1, o_2).

Definition 3 formalizes the conditions for enablement of different types of objects. It defines a function that takes as parameter an object, a state of a BPD execution and a set of "visited objects" V.

Definition 3 (Object Enablement). *Given a BPD and a token count function* tc*, an object o is enabled in the state represented by* tc *iff* enabled(o, tc, ∅) *is true, where:* enabled: $\mathcal{O} \times (\mathcal{F} \to \mathbb{N}) \times (\wp(\mathcal{O}) \to \mathbb{B})$ *is defined as follows:*

$$
\begin{aligned}
&\mathit{enabled}(o_1, tc, V) = \\
&\quad o_1 \in \mathcal{G}^{\mathcal{X}} \wedge \exists o_2 \in \mathit{pred}(o_1) tc((o_2, o_1)) \geq 1 \vee \\
&\quad o_1 \in (\mathcal{A} \cup \mathcal{E}^{\mathcal{I}}) \wedge \ \exists o_2 \in \mathit{pred}(o_1) \setminus \mathcal{G}^{\mathcal{E}} \ tc((o_2, o_1)) \geq 1 \wedge \\
&\qquad\qquad \exists o_2 \in \mathit{pred}(o_1) \cap \mathcal{G}^{\mathcal{E}} \ \exists o_3 \in \mathit{pred}(o_2) \ tc((o_3, o_2)) \geq 1 \vee \\
&\quad o_1 \in \mathcal{G}^{\mathcal{P}} \wedge \forall o_2 \in \mathit{pred}(o_1) \ tc((o_2, o_1)) \geq 1 \vee \\
&\quad o_1 \in \mathcal{G}^{\mathcal{C}} \wedge \exists s \in \mathit{abf}(o_1) \ \forall o_2 \in s \ tc(o_2, o_1) \geq 1 \vee \\
&\quad o_1 \in \mathcal{G}^{\mathcal{I}} \wedge o_1 \notin V \wedge \ \exists o_2 \in \mathit{pred}(o_1) \ tc(o_2, o_1) \geq 1 \wedge \\
&\qquad\qquad \forall o_2 \in \mathit{pred}(o_1) \ tc(o_2, o_1) = 0 \Rightarrow \\
&\qquad\qquad\qquad \neg\exists o_3 \in \mathit{pred}^*(o_2) \ \mathit{enabled}(o_3, tc, V \cup \{o_1\})
\end{aligned}
$$

For objects other than OR-joins, it is straightforward to determine if they are enabled. For example, an exclusive gateway ($\mathcal{G}^{\mathcal{X}}$) is enabled if there is at least one token in at least one of its incoming flows.[2] Activities ($\mathcal{A}$) and intermediate events ($\mathcal{E}^{\mathcal{I}}$) are also enabled if there is at least one token in their incoming flow. An exception to this latter rule occurs if one of the predecessor of the activity or event in question is an event-driven choice gateway. In this case, the activity/event is enabled if there is at least one token in one of the incoming flows of that event-driven choice gateway. A parallel gateway ($\mathcal{G}^{\mathcal{P}}$) requires all incoming flows to carry a token for enablement. For the complex gateway ($\mathcal{G}^{\mathcal{C}}$), a subset of the incoming flows need to all contain at least one token. In the concrete syntax of BPMN, a boolean condition over sequence flows is given to capture under which situations is a complex gateway enabled, i.e. which are the possible subsets of incoming flows that need to contain tokens for the complex gateway to fire. Here, we abstract away from the concrete syntax and we assume the existence of a function abf which given a complex gateway o_1 in a BPD, retrieves the set of possible subsets of predecessors of o_2, such that a token needs to be present in each flow (o_2, o_1) for the complex gateway to be enabled.

For inclusive OR Gateways ($\mathcal{G}^{\mathcal{I}}$), specifically those with more than one incoming flow (i.e. OR-joins), the semantics is more complicated. The informal semantics is such that if any object in the set of predecessors of an OR-join is enabled, the OR-join should wait. If there is at least one token at each of the incoming flows of an OR-join, the OR-join is clearly enabled. Otherwise, it is necessary to explore some or all of the predecessors of the OR-join, to detect whether the OR-join needs to wait for them or not. This may lead to a recursive definition if an OR-join is its own set of predecessors. To avoid an infinite recursion, the definition of enablement keeps track of the set of OR-joins that have been visited. This is the role of parameter V.

The first time an OR-join is visited, its semantics is treated as non-local: the OR-join is enabled if and only if there is at least one token in one of the incoming flows and all predecessors along incoming flows with no tokens are disabled. For

[2] Event-driven choice gateways ($\mathcal{G}^{\mathcal{E}}$) are never enabled. Their presence however determines whether or not the objects that immediately succeed them are enabled.

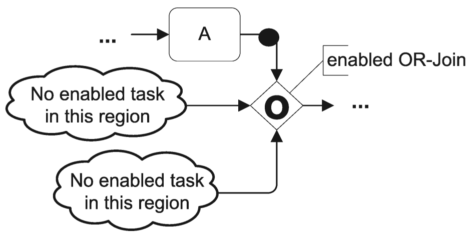

A
enabled OR-Join
No enabled task in this region
No enabled task in this region
O
...

Fig. 4. Checking if a non-visited OR-join is enabled

example, Figure 4 shows an OR-join with three incoming flows. The flow coming from A contains a token (shown as a black dot). To determine if this OR-join is enabled, we inspect the set of predecessors along the other flows (which have no tokens in the current state) and we check that none of them is enabled.

Meanwhile, if an OR-join has already been visited, it is necessarily in partial or full conflict with another OR-join. Indeed, an object o is only added to the set of visited objects V by a recursive call to function *enabled*, with the first parameter of this call being a predecessor of o along an empty flow. So if o has already been visited, it means that there is an OR-join o'_1 such that $o'_1 \in$ pred*$(o_1) \wedge o_1 \in$ pred*(o'_1) and such that function *enabled* has been previously called with o'_1 as first parameter and this call generated a call to *enabled* with o_1 as first parameter.[3] Thus, we have a situation where, to determine if o'_1 is enabled, we ask the question of whether or not o_1 is enabled, and vice-versa as depicted in Figure 2. To resolve this conflict, we treat o_1 as not enabled with respect to o'_1. Accordingly, if $o_1 \in V$, the function evaluates to false.

4 Algorithm for Determining Enablement of an OR-Join

A naive implementation of the formal definition of enablement is computationally expensive. To determine whether an OR-join is enabled, a naive algorithm needs to potentially visit every predecessor of the OR-join (transitive or not), and for each non-visited OR-join in this set of predecessors, it needs to make a recursive call. In the worst-case, each object is a predecessor of each other and then, the number of recursive calls of the enablement function is equal to the number of objects in the BPD minus the number of visited objects. Thus, the complexity of the naive algorithm is captured by the following recursive function: $c(T, X) = (N - X) \times c(N, X - 1)$ where N is the number of objects in the BPD and X is the number of visited objects. Given a BPD with T tasks, the complexity of the function call enabled(o, tc, { }) is thus in the order $O(T!)$.

The problem with the naive algorithm is that each object is examined a repeated number of times, and each time, the same question is asked, namely "is the object enabled?" If the BPD is primarily composed of OR-joins and all these OR-joins are in structural conflict, this leads to a combinatorial explosion.

[3] Here, o_1 and o'_1 may be the same object.

Below we present an algorithm that overcomes this combinatorial explosion by avoiding the recursion. We achieve this by making the following observation: An OR-join for which enablement needs to be determined (say gateway o_1) does not actually need to know whether a preceding OR-join (say o'_1) is enabled or not. What is important is to determine if o_1 must wait for o'_1 assuming that o_1 is not waiting for any other of its precedessors. Indeed, the algorithm will visit all relevant predecessors of o_1, and if it determines that o_1 needs to wait for any of them, the algorithm will return false. Under this assumption gateway o_1 must wait for o'_1 if the following conditions hold:

1. There is at least one token in at least one incoming flow of o'_1.
2. There is at least one token in each incoming flow of o'_1 that is part of a path starting at o_1 and finishing at o'_1.[4]

The first condition is a necessary condition for enablement of an OR-join, and thus it is a necessary condition for o_1 to have to wait for o'_1. The second condition is also necessary. If this condition was false for a given path from o_1 to o'_1, then o_1 and o'_1 are in full conflict, thus entailing that o_1 must not wait for o'_1. Indeed, we are assuming that o_1 is not waiting for any other of its predecessors to fire. We can then conclude that o'_1 is not waiting for any of its predecessors neither (apart from o_1) since all predecessors of o'_1 are also predecessors of o_1. So o_1 and o'_1 are waiting only for one another, and hence they are in full conflict.

The two conditions are also sufficient for o_1 to have to wait for o'_1. Indeed, one of the characteristics of the definition of the OR-join is that if there is a token in one of the incoming flows of this OR-join (o'_1 in this case), this OR-join will eventually fire. Hence, if the first condition is true then o_1 must wait for o'_1 unless there is a possibility of a full conflict between the two gateways. This latter case is excluded if the second condition also holds.

We split the proposed algorithm into two functions: one for determining the enablement of any object except an OR-join, and the other for OR-joins. Accordingly, we first define a function *IsObjectEnabled* that determines if a given object is enabled for a BPD (see Figure 5). This function implicitly takes as input a BPD, but for simplicity, the BPD is not shown as a parameter; instead, the components of the BPD (e.g. $\mathcal{G}^{\mathcal{I}}$) are referred to in the body of the function.

Function call $\mathit{IsObjectEnabled}(o, tc)$ returns true iff object o is enabled in state tc. Unlike the enablement function in Section 3.2, *IsObjectEnabled* does not maintain a set of visited objects, as it is not recursive. This function handles all gateways except the OR-join. Enablement of an OR-join is determined by another function, namely *IsOREnabled* – see Figure 6. The function call $\mathit{IsOREnabled}(o, tc)$ returns true iff OR-join o is enabled in state tc.

The algorithm first checks that the OR-join o has at least one token in at least one of its incoming flows. If so, it iterates over the set of predecessors of the OR-join along an empty incoming flow (i.e. an incoming flow of o that has no

[4] Importantly, if there is no path from o_1 and o'_1, meaning that there is no cycle involving both o_1 and o'_1, this condition is true.

```
FUNCTION IsObjectEnabled($o_1 : \mathcal{O}$, $tc : \mathcal{F} \to \mathbb{N}$): $\mathbb{B}$
 case
   $o_1 \in (\mathcal{G}^{\mathcal{X}} \cup \mathcal{G}^{\mathcal{E}})$:
     return $\exists o_2 \in pred(o_1)\ tc(o_2, o_1) \geq 1$;
   ... (other cases except $o_1 \in \mathcal{G}^{\mathcal{I}}$ treated as per Definition 3)
   $o_1 \in \mathcal{G}^{\mathcal{I}}$:
     return $IsOREnabled(o_1, tc)$;
 end case;
```

Fig. 5. Algorithm to determine whether an object is enabled

```
1   FUNCTION IsOREnabled($o : \mathcal{G}^{\mathcal{I}}, tc : \mathcal{F} \to \mathbb{N}$) : $\mathbb{B}$
2   if $\neg\exists po \in pred(o)\ tc((po, o)) \geq 1$ then
3     return false;
4   else
5     PredAlongEmptyFlows := $\{o' \in \text{pred*}(po) \mid \exists po \in pred(o)\ tc((po, o)) = 0\}$
6     foreach $o' \in$ PredAlongEmptyFlows do
7       if $o' \in \mathcal{G}^{\mathcal{I}} \land \exists po' \in pred(o')\ tc((po', o')) \geq 1 \land$
8          $\forall po' \in pred(o')\ o \in \text{pred*}(po') \Rightarrow tc((po', o')) \geq 1$ then
9         return false
10      else if $o' \notin \mathcal{G}^{\mathcal{I}} \land$ IsObjectEnabled($o'$, $tc$) then
11        return false;
12      end if
13    end foreach;
14  return true;
```

Fig. 6. Optimized algorithm to determine if an OR-join is enabled.

token). For each of these predecessor objects, the function returns false if either the object in question is an OR-join and it satisfies the above two conditions (cf. lines 7 and 8 respectively), or it is not an OR-join and it is enabled as determined by function IsObjectEnabled (cf. line 10). If all the predecessors of the OR-join are visited and none satisfies any of these conditions, it means the OR-join should not wait for anything and thus the function returns true.

To analyze the algorithm's complexity, we first observe that function pred* involves computing a transitive closure which has a complexity of $O(|V| + |E|)$, E being the set of edges (flows in the BPD) and V the set of vertices (i.e. objects). Thus, the complexity of one invocation to this function is $O(N)$ where N is the total number of elements in the BPD. After computing the set of predecessors along each empty incoming flow of o, the algorithm iterates over this set of predecessors, which in the worst case includes all objects in the model except for end events. If one of these predecessors (o') is itself an OR-join, the algorithm checks if there is a path between o and o'. This latter step involves invoking function pred* for each incoming flow of o'. We can thus bound the worst-case complexity of the algorithm by $O(N^2)$, as function pred* is potentially invoked for each element in the model, and each invocation has a cost of $O(N)$.

A substantial reduction in time complexity can be achieved by computing at design-time the set of predecessors of each object in the model, and storing the result in such a way that the set of predecessors of an object can be retrieved in constant time, e.g. using a hash table where the keys are objects in the BPD and the values are sets of predecessors. The size of this data structure is $O(T \times N)$, where T is the number of objects in the model (excluding flows). Once the data structure is materialized and the invocations to pred* are replaced by constant-time lookups, the complexity of the algorithm is reduced to $O(N)$.

5 Incremental Evaluation

Since the complexity of evaluating enablement for OR-joins is still higher than that for other gateways, it is desirable to further optimize the evaluation procedure. We therefore reuse the result of the evaluation of an OR-join's enablement in one state, when determining enablement of this OR-join in the next state. In other words, we would like to materialize the results of evaluating the enablement of each OR-join, so that after a state change (e.g. after an enabled object in the BPD fires), we only need to examine objects affected by the change. We call this *incremental evaluation.*

We assume a state change is represented as a set of flows in which tokens have been either added or removed. We capture all information pertaining to the enablement of each OR-join in a global (hash) table, namely *mustWaitFor*, which associates to each OR-join in the model, the set of predecessors for which this OR-join would have to wait if it was partially enabled, i.e. if it had at least one token in one of its incoming flows. For convenience, we write $mustWaitFor[o]$ to refer to the entry in this table corresponding to object o, i.e. the set of predecessors that o must wait for as evaluated in the state prior to the change.

The incremental evaluation function is algorithmically described in Figure 7. The function takes as input an OR-join, the current state of the execution (after the change), and the state change Δ. The function should be called each time a state change occurs.

The first part of the algorithm (lines 2-11) updates the set $mustWaitFor[o]$. For each predecessor o' of o such that one of the incoming flows of o' has changed, the algorithm evaluates the new state of enablement of o' and, if necessary, it adds or removes o' from set $mustWaitFor[o]$. For this purpose, we reuse lines 7 and 8 of the *IsOrEnabled* algorithm, as well as function *IsObjectEnabled*, but we only call this latter function for objects other than OR-joins. In this "updating" phase of the algorithm, we consider predecessors of o along flows with no tokens as well as predecessors of o along flows that already contain tokens. The rationale is that all changes have to be accounted for, even if they do not have an immediate influence on the enablement of the OR-join o. Indeed, if o already has a token in one of its incoming flows, it will eventually fire, and when this happens, tokens will be removed from some of its incoming flows. As a result, some incoming flows may switch from having one token to having no token, and previously irrelevant changes may become relevant again.

```
1  FUNCTION IsOREnabledInc($o : \mathcal{G}^{\mathcal{I}}, tc : \mathcal{F} \rightarrow \mathbb{N}, \Delta : \wp(\mathcal{O} \times \mathcal{O})) : \mathbb{B}$
2   foreach $(o'', o') \in \Delta$ where $o' \in \text{pred}^*(o) \setminus \{o\}$ do
3     if $o' \in \mathcal{G}^{\mathcal{I}} \wedge \exists po' \in pred(o')\ tc(po', o') \geq 1 \wedge$
4        $\forall w \in pred(o')\ o \in \text{pred}^*(w) \Rightarrow tc(w, o') \geq 1$ then
5       $mustWaitFor[o] := mustWaitFor[o] \cup \{o'\}$;
6     else if $o' \notin \mathcal{G}^{\mathcal{I}} \wedge IsObjectEnabled(o', tc)$ then
7       $mustWaitFor[o] := mustWaitFor[o] \cup \{o'\}$;
8     else
9       $mustWaitFor[o] := mustWaitFor[o] \setminus \{o'\}$;
10     end if;
11   end foreach;
12   if $\exists po \in pred(o)\ tc(po, o) \geq 1$ then
13     foreach $po \in pred(o)$ where $tc(po, o) = 0$ do
14       if $\exists o' \in \text{pred}^*(po)\ o' \in mustWaitFor[o]$ then return false;
15     end foreach
16     return true;
17   else
18     return false;
19   end if
```

Fig. 7. Algorithm for incrementally evaluating OR-join enablement.

Once all changes are accounted for, the status of the OR-join has to be re-evaluated (lines 12-19). Here, all predecessors along empty flows are checked. If at least one of them object is in the set *mustWaitFor*$[o]$, then o is not enabled.

6 Related Work

An assessment of fourteen state of the art commercial offerings in [15] revealed that only a handful of them support the OR-join construct without imposing syntactic restrictions. Many other languages support the OR-join but only in restricted settings. For example, in BPEL [9] it is only possible to define an OR-join in the context of acyclic networks of activities connected through so-called *control links*. In such restricted settings, the semantics of the OR-join becomes easier to define. Similarly, workflow management systems like InConcert, eProcess, and WebSphere MQ Workflow have avoided the problems related to defining the OR-join semantics by introducing syntactic restrictions. Eastman supports an OR-join with non-local semantics, but it is acknowledged in the Eastman manual that the use of OR-joins may result in poor performance [8].

There are several proposals to formally define a semantics for the OR-join in Event Process Chains (EPCs) without introducing syntactic restrictions. In van der Aalst et al [1], the problems with the OR-join semantics in EPCs, especially that of vicious circles, are highlighted. It is suggested that any formal OR-join semantics will impose some restrictions or will deviate from the informal semantics to some extent. This leads to a proposal by Kindler [10,11] to define a non-local semantics in EPCs (including the OR-join) in terms of a pair of transition relations using techniques from fixed point theory. Kindler's semantics can be considered as the most general and precise semantics of the OR-join, but as acknowledged by the author in subsequent papers, such fixed-point techniques are "very inefficient and intractable in practise" [5,6]

To address this computational complexity hurdle, Kindler proposed the use of binary decision diagrams (BDDs) to represent large sets of states and large transition relations in a compact manner [11]. Cuntz et al [4] later proposed a concrete method to calculate these transition systems using Kleene's fix-point theorem and techniques from symbolic model checking, instead of using a fixed-point iteration as in Kindler's original proposal. Specifically, Cuntz et al outline a technique for calculating the semantics of EPCs that combines a forward construction of the transition system with a backwards marking algorithm. Unfortunately, this optimized algorithm is not complete, meaning that it does not work on all EPC models. Also, despite these optimizations, the complexity of their approach is still proportional to the size of the transition system and hence, the approach only works for EPCs with small state spaces. In contrast, the OR-join semantics we have put forward can be evaluated in linear time on the number of elements in the process model, making it more scalable.

Mendling et al. [12] explore another approach to formally define the semantics of OR-joins in syntactically correct EPCs. This new semantics is inspired by the semantics of the OR-join in BPEL, but it can be applied to process models with arbitrary cycles. The semantics of Mendling et al. relies on the notion of state (i.e., tokens attached to arcs as in our approach) in combination with a notion of context (i.e., special tokens indicating if a given arc is in a "wait" or in a "dead" status). The context of an input arc to an OR-join is then used to determine whether an OR-join should be enabled at a given state. The evaluation of this OR-join semantics requires that both "normal" tokens, as well as "wait status" and "dead status" tokens are propagated during the execution of the process model. In contrast, our approach does not require the propagation of additional types of tokens. Also, in the semantics proposed by Mendling et al., an OR-join may "wait forever" if there is a deadlock preceding an OR-join. Specifically, if an AND-join connector "upstream" can not propagate tokens due to a deadlock, an OR-join connector "downstream" will continue to wait for a token forever. In contrast, our semantics will detect the deadlock situation upstream and the OR-join will be enabled rather than waiting for a token that will never arrive. In other words, the semantics of the OR-join presented in this paper detects deadlock situation and allows the OR-join to fire despite such deadlocks.

Wynn [19] proposed a general OR-join semantics for the YAWL workflow language that takes into account the "cancellation region" construct supported by this language. A concrete algorithm based on the backwards coverability of reset nets together with two optimization techniques are given to support the implementation in the YAWL workflow environment [18]. In line with Kindler's semantics, the OR-join semantics in YAWL is far-sighted: As soon as it is possible to conclude that an unmarked branch leading into an OR-join will not be reached, the OR-join will detect this situation and not wait anymore for tokens along that branch. In contrast, the semantics proposed in this paper is more short-sighted: only when no state change whatsoever can occur among the set of predecessors leading into an incoming branch of an OR-join, will the OR-join fire. This choice constitutes a tradeoff between precision (i.e. how early do we detect that an OR-join can fire) and the efficiency of evaluating the enablement of an OR-join. Another difference is that the semantics proposed in this paper does not lead to deadlocks in the presence of vicious circles, while the YAWL semantics does. A major issue addressed by the OR-join semantics of YAWL is that of dealing with the notion of "cancellation feature". This is similar to the notion of "exception handler" in BPMN. However, in YAWL it is possible to cancel certain parts of a (sub)-process without canceling the entire (sub)-process. This leads to an interference between the OR-join behavior and the cancellation behavior. In BPMN, this type of cancellation behavior is impossible. Exception handling behavior in BPMN is attached to a sub-process and when an exception occurs, all the tokens from this sub-process are removed at once (and therefore all the OR-joins inside the sub-process are disabled). Therefore, the OR-join construct and the exception handling construct in BPMN do not interfere with one another.

In conclusion, our major contribution with respect to previous related work has been to define a semantics of the OR-join which has a linear computational complexity and can be evaluated incrementally, while at the same time not imposing syntactic restrictions on the process models and maintaining the desirable properties of a non-local OR-join semantics.

7 Conclusion

We have proposed a formalization of the enablement rules for a subset of BPMN objects, including the OR-join gateway, with the following characteristics:

- It does not impose restrictions on the topology of the process models, other than the minimal restrictions imposed by the syntax of BPMN itself.
- It has a relatively low computational complexity: determining if an OR-join is enabled can be computed in $O(N^2)$ where N is the total number of elements (objects + flows) in the model. This complexity can be reduced to $O(N)$ after materializing a data structure of size $O(N^2)$ at design-time. In addition, the semantics lends itself to an incremental evaluation mode.
- It does not generate deadlocks in the presence of cycles in the model involving multiple OR-joins.

The proposed OR-join semantics can be labeled as *operational* as it captures how an execution of a process model moves from one state to another. For static analysis purposes, it is desirable to have a semantics defined by translation of BPMN to a formalism (e.g. Petri nets) for which static analysis tools are available. In future, we plan to extend our mapping from BPMN to Petri nets [7] with the ability to deal with OR-joins. This would involve defining rules to transform BPMN models that contain OR-joins, into models that do not, since Petri nets do not directly support this construct. As a by-product, such transformation rules could be useful in building simulation engines for BPMN, as they would replace all OR-joins with constructs that are easier to simulate.

Acknowledgments. We thank Alistair Barros for his valuable comments. The first author is supported by a fellowship funded by Queensland Government and SAP. The second author conducted this work while at SAP Research.

References

1. van der Aalst, W.M.P., Desel, J., Kindler, E.: On the Semantics of EPCs: A Vicious Circle. In: Rump, M., Nüttgens, F.J. (eds.) Proceedings of the EPK 2002: Business Process Management using EPCs, Trier, Germany, pp. 71–80. Gesellschaft für Informatik (2002)
2. van der Aalst, W.M.P., ter Hofstede, A.H.M, Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases 14, 5–51 (2003)
3. van Breugel, F., Koshkina, M.: Models and verification of BPEL. Working paper (September 2006), http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf
4. Cuntz, N., Freiheit, J., Kindler, E.: On the semantics of EPCs: Faster calculation for EPCs with small state spaces. In: Nüttgens, F.J., Rump, M. (eds.) Proceedings of EPK 2005, Hamburg, pp. 7–23 (December 2005)
5. Cuntz, N., Kindler, E.: On the semantics of EPCs: Efficient calculation and simulation. In: Nüttgens, F.J., Rump, M. (eds.) Proceedings of EPK 2004, pp. 7–26 (October 2004)
6. Cuntz, N., Kindler, E.: On the semantics of EPCs: Efficient calculation and simulation (Extended Abstract). In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 398–403. Springer, Heidelberg (2005)
7. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and automated analysis of BPMN process models. Preprint 5969, Queensland University of Technology (January 2007), https://eprints.qut.edu.au/archive/00005969
8. Eastman Software. RouteBuilder Tool User's Guide. Eastman Software, Inc, Billerica, MA, USA (1998)
9. Jordan, D., Evdemon, J. (eds.): Web Services Business Process Execution Language Version 2.0. OASIS WS-BPEL TC (2005), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
10. Kindler, E.: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In: Desel, J., Pernici, B., Weske, M. (eds.) BPM 2004. LNCS, vol. 3080, pp. 82–97. Springer, Heidelberg (2004)
11. Kindler, E.: On the Semantics of EPCs: Resolving the Vicious Circle. Data and Knowledge Engineering 56(1), 23–40 (2006)
12. Mendling, J., van der Aalst, W.M.P.: Formalization and Verification of EPCs with OR-Joins based on State and Context. In: CAiSE 2007. Proceedings of the 19th International Conference on Advanced Information Systems Engineering, Trondheim, Norway, Springer, Heidelberg (to appear, 2007)
13. OMG. Business Process Modeling Notation (BPMN) Version 1.0. OMG Final Adopted Specification. OMG (February 2006), http://www.bpmn.org/

14. Reis, S., Metzger, A., Pohl, K.: Integration testing in software product line engineering. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, Springer, Heidelberg (2007)
15. Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mulyar, N.: Workflow control flow patterns: A revised view. BPMCenter Technical report BPM-06-22, BPMCenter.org (2006)
16. Silver, B.: The 2006 BPMS Report: Understanding and Evaluating BPM Suites (2006), http://www.bpminstitute.org/bpmsreport.html
17. Wong, P.Y.H., Gibbons, J.: A process semantics for BPMN. Preprint, Oxford University Computing Laboratory (March 2007), http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmn_extended.pdf
18. Wynn, M.T.: Semantics, Verification, and Implementation of Workflows with Cancellation Regions and OR-joins. PhD Thesis, Faculty of Information Technology, Queensland University of Technology (November 2006)
19. Wynn, M.T., Edmond, D., van der Aalst, W.M.P., ter Hofstede, A.H.M: Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 423–443. Springer, Heidelberg (2005)
20. Wynn, M.T., Edmond, D., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 423–443. Springer, Heidelberg (2005)

Pattern-Based Design and Validation of Business Process Compliance

Kioumars Namiri[1] and Nenad Stojanovic[2]

[1] SAP Research Center CEC Karlsruhe, SAP AG, Vincenz-Prießnitz-Str.1
76131 Karlsruhe, Germany
Kioumars.Namiri@sap.com
[2] FZI Karlsruhe, Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
Nenad.Stojanovic@fzi.de

Abstract. In this paper we present a novel approach for the modeling and implementation of Internal Controls in Business Processes. The approach is based on the formal modeling of Internal Controls in the validation process under the usage of frequently recurring control patterns. The main idea is the introduction of a semantic layer in which the process instances are interpreted according to an independently designed set of controls. This ensures separation of business and control objectives in a Business Process. A prototypical implementation of the approach is presented.

Keywords: BPM, Regulatory Compliance, Internal Controls, Patterns.

1 Introduction

The advent of regulatory compliance requirements in the area of Internal Controls such as Sarbanes Oxley Act 2002 (SOX) [1] requires the implementation of an effective Internal Controls system in enterprises as a management responsibility. In this context COSO (Committee of Sponsoring Organizations of the Treadway Commission) has proposed an integrated framework [2], which is recognized by regulation bodies and auditors as a de facto standard for realizing the Internal Controls System. COSO defines the Internal Controls as a "process" designed to provide reasonable assurance regarding the achievement of objectives in effectiveness and efficiency of operations, reliability of financial reporting, and compliance with applicable laws and regulations. The realization and effectiveness of this process involves different roles: Internal Auditing consultants/Compliance experts, external regulation bodies, Business Process experts (including system developers/technical consultants). Each of these roles has a different view on the enterprise, uses different terminology for speaking about the same domain and requires specific system support. This is one of the main issues why the introduction and operations of Internal Controls compliance (e.g. SOX 404) is considered to be expensive and time consuming [3].

This paper presents a semantic-based approach for managing above mentioned complexity in the realization of the compliance process. In the nutshell of the approach is the patter-based abstraction layer that separates business process and compliance management. Two sets of compliance patterns are introduced: a set of high level control patterns, which represent the way Compliance/Business Process experts communicate about the compliance domain and a set of system level control patterns based on the property specification pattern system proposed originally by Dwyer at al [5]. Each pattern on its abstraction level should accordingly give software and compliance/business practitioners access to specify and design the compliance requirements. We are mostly concerned with automation of the so called Application Controls (AC), which control Business Processes to support financial control objectives and to prevent or detect unauthorized transactions. However, the approach provides a general framework that can be applied with respect to any other compliance domain using BPM technology.

The paper is organized as follows: We start with a motivating scenario. In the third section we introduce the domain model of Internal Controls/SOX compliance, which we base on our analysis of the COSO framework. In the fourth section we present the patterns for compliance controls and their usage. In section five we introduce the approach built on top of the domain model and the patterns introduced previously whereas the sixth section explains its implementation architecture. Related literature is discussed in section seven. Concluding remarks and some future research questions are given in the last section.

2 Motivating Scenario

We use the Purchase-To-Pay (P2P) Process delivered by an ERP product as an example. The process starts by creating the request for a Purchase Order (PO) and ends when the payment of that PO is recorded in Accounting. An excerpt of P2P is illustrated in **Fig. 1**.

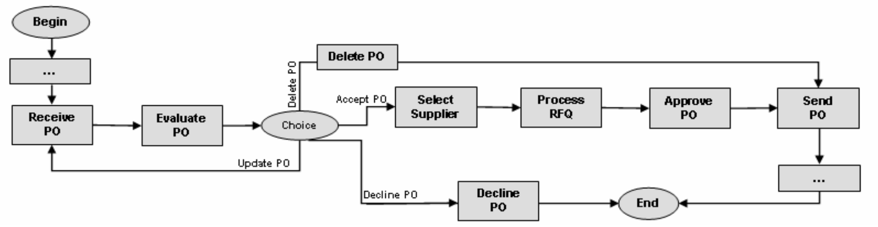


Fig. 1. Purchase-To-Pay (P2P) Process: an excerpt

The Internal Controls compliance of P2P depends on an enterprise specific risk assessment. Table 1 shows an excerpt of the risk assessment carried out by Compliance experts of two different enterprises. It shows their different control objectives, risks, and controls on the same standard P2P Process.

We have identified that there exist frequently defined patterns of controls on Business Processes at different enterprises. Taking the perspective of an ERP vendor, providing this set of patterns in a repository where a certain pattern can be selected, instantiated to a real control, and applied on Business Processes by their customers brings a higher level of system and component reusability for the ERP/BP products. Taking the perspective of a customer company building their compliance on top of such a pattern repository can reduce the required domain specific knowledge in compliance projects. Therefore, the process models become nowadays too complicated, not readable and manageable when they are directly, i.e. manually enriched with the necessary compliance controls. In the rest of the text we present an approach that copes with this kind of complexity.

Table 1. Risk assessment on Purchase-To-Pay (P2P) Process for two different enterprises

Control Objective	Risk	Control
Enterprise A: Prevent unauthorized use	Unauthorized creation of POs and payments for not existing suppliers	1) POs for material types which have not been ordered during last year and an amount higher than 5000 $ must be double approved by two different purchasing clerks (Second Set of Eyes Control - SSE). 2) Segregation of Duties (SoD) on PO Creation and PO Approvals with an amount higher than 5000 $.
Enterprise B: React flexible on changes in the supplier market	Dependancy on one single supplier in the market	Minimum Number of Suppliers is 2 for material type 5: Keep at least two contracted Suppliers in your Supplier Relationship Management (SRM) System for the given material type.

3 Domain Model for Internal Controls Compliance

One of the main issues in the separation of the business and control objectives of a Business Process is that business objectives and control objectives for a Business Process have different life cycles and stakeholders. Figure 2 illustrates how we see the relationship between BPM and Internal Controls Management [11]: The design of a control should control the way a Business Process is executed. A (re)design of a Business Process causes an update of risk assessment on a Business Process, which may lead to a new/updated set of controls including new tests. The Business Process monitoring and validation techniques may be used to assess the effective design of controls and their operations and can serve as an input to Compliance certification.

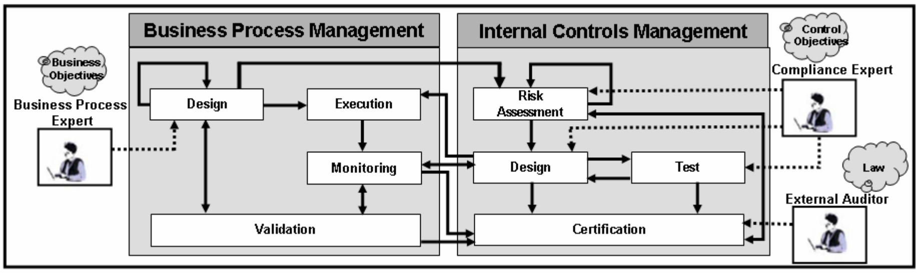


Fig. 2. Relations between BPM and Internal Controls Management

3.1 Roles Involved

We distinguish three roles involved in Business Process Compliance with the following interests/expertise:

Business Process Expert: A Business Process expert knows how to configure and maintain the processes having business objectives (goals) in mind. The business objective for, e.g. a purchasing process, is simply to set up a process in which internal orders created can be processed and sent to suppliers, to receive the ordered goods, and to pay the supplier invoices. It is obvious that in large scale ERP systems this role is carried out by different persons, even different organizational units. This group of persons in an enterprise has no or little knowledge about regulations and compliance requirements, but very detailed knowledge on how a process is implemented.

Compliance Expert: The auditing consultants are Compliance/SOX experts and have detailed knowledge about the regulatory requirements. They have no or little knowledge about the realization of Business Processes in an enterprise. Their main task is rather to define and monitor the necessary controls according to the risk assessment and to notify other entities in the enterprise in case of control violations. They do not define how to bring a process in a compliant state because this is the task of Business Process experts.

External Auditors: External auditors are regulation bodies or official firms who certify the **design** and **effectiveness** of the Internal Controls system in an enterprise on a periodical basis. The external auditors are out of scope in this work.

3.2 Interplay of the Entities in the Domain Model

Based on this view, we introduce a set of models for implementing the Internal Controls process. We are concerned with providing a cooperative environment for Business Process experts and Compliance experts to achieve their necessary tasks regarding Business Process compliance as discussed above.

The entities and their relations to each other provide the terminology used to formulate logical statements representing the controls constraining the behavior of a Business Process.

In the following we enrich the entities resulted from our analysis of (mainly not IT-related) COSO by additional entities. These additional entities will enable the model to serve to us as an operational basis for our approach later on. Only those parts of COSO necessary for understanding our approach are presented Figure 3. It shows the upper domain model of required entities for the Internal Controls process.

Each *Risk* is assigned to those *Business Processes* that affect a *Significant Account*. A risk is assessed according to its *Control Objectives*. A Business Process may contain multiple risks (or not) and the same risk may occur in different Business Processes. *Application Control* (AC) is a subtype of the entity *Control*. We further introduce *Company Level Controls* and *IT Controls*. Company Level Controls are controls covering the “Control Environment”-component in COSO, which requires the management to create a social environment in an enterprise where effective Internal Controls are to be achieved. This includes controlling the management as well. IT Controls are controls regarding the realization and operation of IT projects and systems. IT Controls and Company level controls are out of scope for this work.

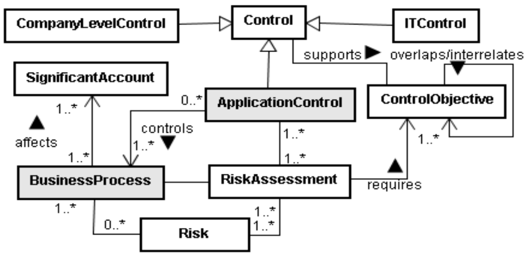


Fig. 3. Upper domain model for Internal Controls compliance

In the following we discuss those parts of the model that are relevant for our approach in detail: *Business Process* and *Application Control* (we simply use the term control).

3.2.1 Control - Business Process Model

Below we further detail the relationship between a *Business Process* and a *Control* as shown in Figure 4.

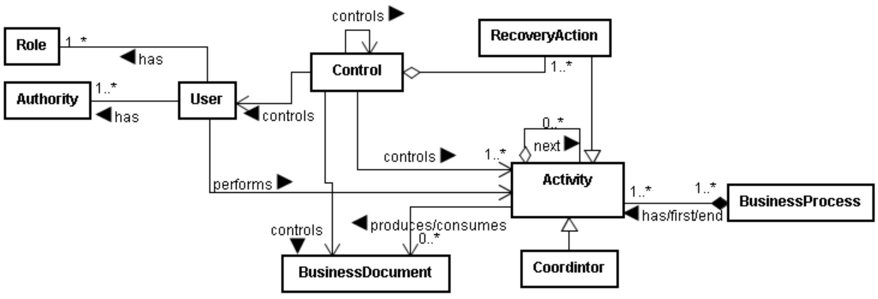


Fig. 4. Relationship between a control and a Business Process

We interpret a business process according to [7], where it is defined as a set of logically related tasks (or activities) to achieve a defined business outcome. An *Activity* can be aggregated by other activities. A special kind of activity is the *Coordinator* Activity (such as switch/fork/join etc.), which defines the behavior of the flow in a business process known from workflow modeling. An *Activity* consumes and produces *Business Documents* (such as *Purchase Order, Invoice etc.*). And finally an Activity is performed by a User (human or computer). Thus a control influences different dimensions of the way a business process is enacted, namely:

- The execution order and occurrence of its activities (including their state changes and attributes such as its duration etc.)
- The Business Documents involved (including their state changes and attributes such as amount etc.) and
- The users performing an activity (including their roles and authorities).

For each *control* at least one *Recovery Action* must have been designed, which reacts on the violation of a control. The nature of the recovery action depends on the current role of the person involved in the business process compliance: The Compliance expert or the Business Process expert. We detail this in the next subsection.

3.2.2 Role Based Recovery Action Model

As a control is originally defined by a Compliance expert in an enterprise, his main objective is to design the control and to monitor its effectiveness. As said before, a Compliance expert has no or little knowledge about the implementation of a Business Process. The detailed knowledge on how to bring a Business Process model/instance into a compliant form/state is the task of a Business Process expert.

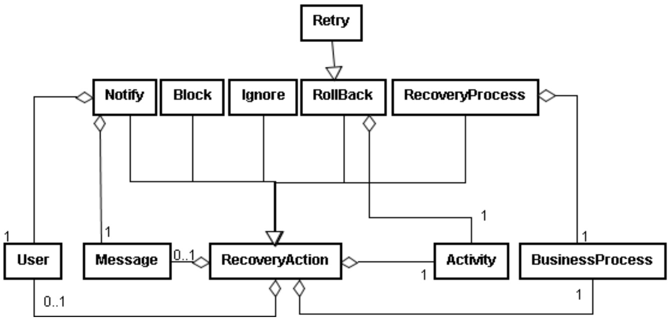


Fig. 5. Recovery Actions on Control Violation

Our model for Business Process compliance recognizes this fact by introducing a role based recovery action model (Figure 5):

In the following we explain the different types of possible recovery actions:

Ignore: The control violation is ignored.

Block: The current instance of the BP, which generated a control violation, is blocked.

Notify (User, Message): A notification message for the specified user *User* is created with the given message *Message*.

Retry: The activity that generated the violation is retried again.

Rollback (Activity): The current instance of the BP that generated the control violation is rolled back to the given activity *Activity.*

Recover (RecoveryProcess): A previously designed recovery process *RecoveryProcess* is instantiated parallel to the current instance of the original BP that generated the control violation. The recovery process itself is an autonomous Business Process.

Please note that a combination of the above listed recovery actions is also possible such as *Retry & Notify etc.*

In case of a control violation a Compliance expert defines the recovery actions as a minimal set of actions regarding the Business Process logic. The decision on recovery action selection in a certain control design is up to the Compliance expert. The decision depends on the enterprise specific risk assessment, which may vary from enterprise to enterprise for the same kind of control. After the control is initially designed with a recovery action by a Compliance expert, it is stored in a control repository. The corresponding Business Process expert is notified about the creation of a new control. The Business Process expert now has the possibility to review and edit the recovery actions for that control, that were originally designed by the Compliance expert.

Based on the recovery action that was originally selected by the Compliance expert, only a limited set of recovery actions is allowed to be selected by the Business Process expert. The valid combination of recovery actions set by the Compliance expert and Business Process expert follows these basic rules:

- A control violation always requires a reaction, particularly a single *Ignore* is never allowed, since the existence of a control with such a recovery model makes that control meaningless.
- The recovery action designed by a Business Process expert is never allowed to "weaken" the original recovery action designed by the Compliance Expert. For instance if an Compliance expert requires a *Block & Notify* on a Business Process instance in case of a certain control violation, the Business Process expert is not allowed to redesign the recovery of a control to only *Notify*.

3.2.3 Controlled Entities

We can see that in the domain model of Business Process compliance for Internal Controls, we have four different types of first class entities: activities, business documents, users, and the controls.

We refer to these four different entities as Controlled Entities (CE) in a Business Process. Their relationship is obvious: An activity performed by a user may consume business documents, may produce new business documents or may change the state of an already existing business document or a user itself. A control exists to assure that a condition defined for one of the other three CEs is effective in a Business Process.

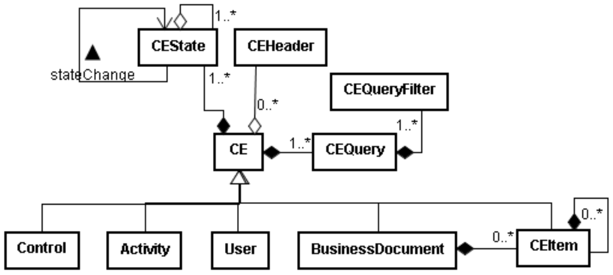


Fig. 6. Composition of Controlled Entity - CE

We consider a control as a controlled entity in a Business Process because the effectiveness of a control should impact the execution of a Business Process. This means basically that if a control is not effective, i.e. its violation has no implications on a Business Process, the enterprise runs the risk of not being compliant. Thus the main tasks of Compliance experts include not only to design the controls but also to assure their effectiveness.

CEs have dependent artifacts in common in their structural composition as visualized in Figure 6. The concept of these artifacts will serve us as a basis for implementing the controls in Business Processes.

A CE may have additional Meta data information (*CEHeader*) specifying an instance of that CE in more detail. Each instance of a CE has a current state (*CEState*) and a set of valid state changes, which are caused by activities executed on an instance of that CE. The item (*CEItem*) of a CE Business Document represents all sub parts of that entity (For instance a PurchaseOrder PO may contain several items for different material types as sub orders). The item can be a CE itself and it may consist of other sub items. The query of a CE (*CEQuery*) determines the number of all instances of that CE according to a given filter (*CEQueryFilter*).

Example: A Query for all POs *POQuery* with a filter *POQueryFilter*

- *approved* POs in
- the *period of last quarter* and
- for a certain *supplier "XYZ"*

will return the number of all PO instances satisfying the given filter criteria.

4 Control Patterns

In the following we introduce two different sets of patterns, which we call high level and system level control patterns. They basically represent the same thing on different abstraction levels in a domain, namely frequently recurring/defined patterns of controls on Business Processes. The high level control patterns provide the basis for the terminology in which the Compliance experts communicate about the domain. We have determined the presented set of high level control patterns empirically by analyzing different kinds of typical ERP Business Processes (Purchasing, Sales and Human Resource Management and all belonging side processes such as Goods Return, Payment, Dunning, etc.). Here we have grouped typical control categories that are defined on those Business Processes at different enterprises built on top of a provided set of process reference models inside an ERP Product.

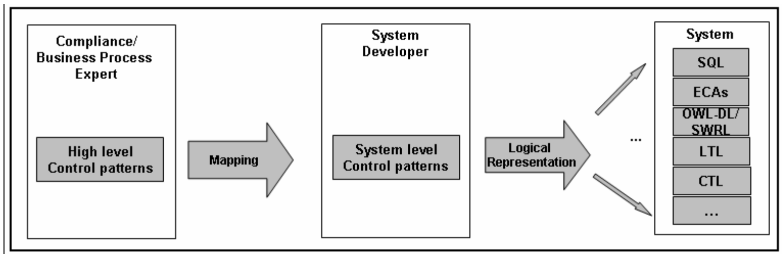

Compliance/ Business Process Expert
High level Control patterns
Mapping
System Developer
System level Control patterns
Logical Representation
...
System
SQL
ECAs
OWL-DL/ SWRL
LTL
CTL
...

Fig. 7. From a High level Control pattern to its technical Representation in a System

The system level control patterns represent a more technical view on the controls and their introduction is aimed to facilitate the use of formal methods by system developers/technical personnel having the task of implementing the controls in ERP/BPM Systems. The system level patterns themselves are generic in their nature in that way that they are not bound to the usage of certain formal logics. Each development team can select its favorite and suitable technical representation of the system level control patterns which can vary from database-oriented/SQL to different temporal logics such as LTL or CTL (see Figure 7).

4.1 High Level Control Patterns

In Figure 8 we expose different categories of control patterns on Business Processes and give a brief description for each pattern category type without going into details.

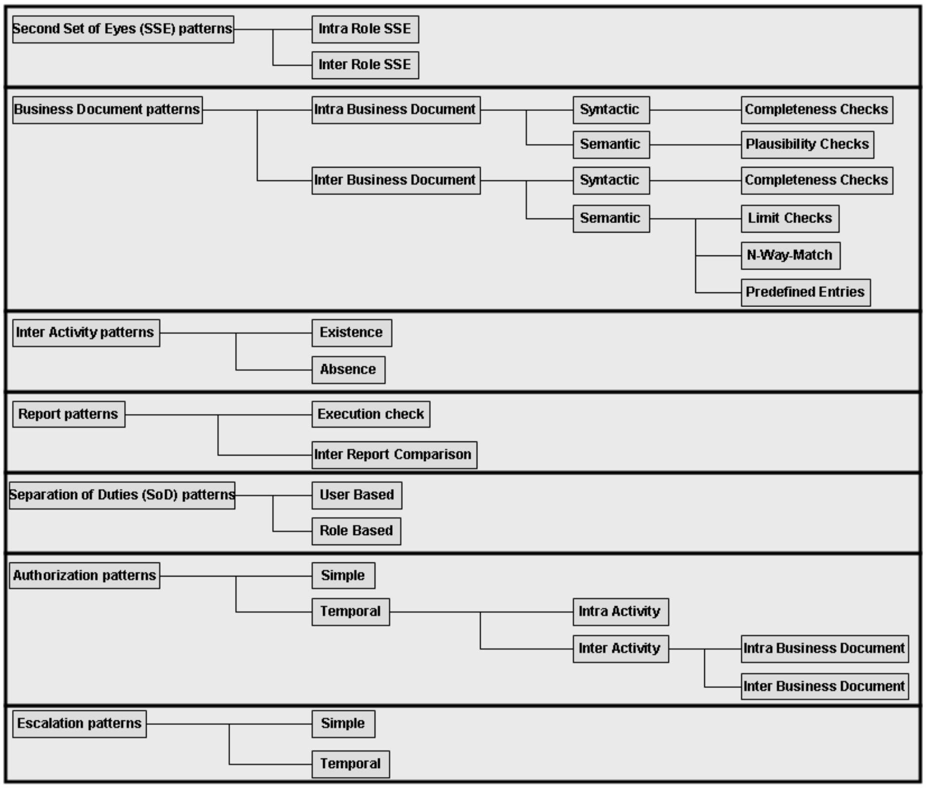


Fig. 8. High level Control Patterns

SSE Patterns: We already mentioned this kind of control patterns briefly in the scenario section, which basically requires the SSE-principle on certain transactions. Here we add the comment that a control demanding a “higher number of eyes” would also be possible and would fall into this category as well.

Business Document Control Patterns: Here the syntax and semantics in and between different business documents are subject to the controls.

Inter Activity Control Patterns: The controls satisfying these patterns require that certain activities occur (or are absent) if certain set of other activities occur in a Business Process (or a side process).

Report Patterns: Reports are collected based on attributes on certain types of activities and business documents in an enterprise during a certain period, e.g. monthly turnover reports. The purpose of report control patterns is not the definition of a report, but rather to control that a report has been generated respectively reports are compared to each other as required in the control.

SoD Patterns: In order to minimize fraud or misusage it is required that an activity is divided into sub activities and each sub activity is executed by different users or roles.

Authorization Patterns: These controls limit users/roles access to CEs.

Escalation Patterns: In case that detected controls are ignored by the responsible users, this fact can/has to be escalated to responsible entities in the enterprise.

Each high level control pattern is specified by following attributes

- Name of the pattern
- (optional) A (nested) list of super type categories of the given pattern, in order to identify the pattern
- Pattern Description: The aim of the control pattern is described.
- Subjected CE: The CE type, which is subject to the control
- Objected CE(s): The CE(s) which are required to design the control
- Control Trigger: the definition of conditions that make a control to be triggered.
- Related to: (optional) the possible dependency links between different types of control patterns. Typically SoD and SSE patterns are related to Authorization patterns and Escalation patterns to all other types of control patterns.
- Example: (optional) a concrete control in P2P process that follows the given pattern

Below we give an example for the specification of the pattern "N-Way-Match" including its description:

- **Description:** certain fields in header and items of different business document types belonging to the same Business Process instance must match each other
- **Subjected CE:** Business Document
- **Objected CE:** Business Document, Activity
- **Related to:** -
- **Control Trigger:** State change of an Activity or Business Document
- **Example:** 3-way match control on PO, Invoice, and Delivery of Business Documents of a P2P process instance if the supplier identification is identical.

4.2 System Level Control Patterns

System level patterns are used to represent the technical representation of a high level compliance pattern. Each high level control pattern corresponds to a system level pattern, which is described by a Control Strategy:

A *Control Strategy* defines the way a control monitors the behavior of one or controlled entities inside a Business Process (Figure 9). In order to become active a *control* requires to be triggered according to the *state* of the process parameters in a *scope*. We defined the two elements of a control strategy *scope* and *pattern* based conceptually on the work done by Dwyer et al [5]. Although their patterns are mainly used for defining formal requirements on program specifications, they can be applied to Internal Controls compliance and the monitoring requirements there. For a detailed description of the scopes and patterns and their semantics please refer to [5].

We have extended the Dwyer patterns by an entity called *CECondition*, which represents a constraint on one or more CEs. This extension is necessary in order to reflect special conditions in the subjected and objected CEs (including their queries and items in case of business documents).

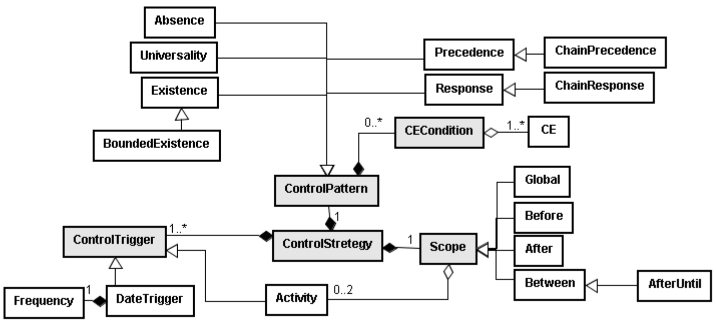


Fig. 9. A Semi-formalization of the control implementation through patterns

Example: Recall the first control on the P2P Process of enterprise A given in the scenario section. This is an "Intra Role SSE" Pattern, which means that it is sufficient that each approver belongs to the same role and can be mapped to the following system level control strategy:

- *ControlTrigger* = Activity "*SelectSupplier*"
- *Scope* = *Between* the activity "*SelectSupplier*" and activity "*SendPO*"
- *Control Pattern* = *Bounded Existence* of *n=2* on CE "*ApprovePO*"-Activity
- *CEConditions:*
  - o *POHeader.amount > 5000$*
  - o *ApprovePO*$_1$*.User.Role = "Purchasing Clerk"*
  - o *ApprovePO*$_2$*.User.Role = "Purchasing Clerk"*
  - o $ApprovePO_1.User.Id \neq ApprovePO_2.User.Id$
  - o $\forall t_i, \forall POItem_i \in \{PO.POItems), POQueryFilter_i = POItem_i.lastOrderDate$ $t_i = POQuery(POQueryFiliter) \mid t_i > 1\ year$

5 The Approach

In order to realize the separation of the business and control objectives presented in Figure 2, our approach introduces another layer above the Business Process model. This layer is called “SemanticMirror”. According to the assessed risks, a set of Controls is defined on that layer. Finally, by executing a Business Process, the semantic process layer will be continually updated with information needed for the evaluation of defined controls in order to ensure that compliance tests will pass. The approach spans over three phases: Control Design phase, Recovery Action Design phase, and Business Process Execution phase. The first two phases each have a sub phase, which we call Business Process Model Adaptation.

5.1 Phase 1 - Control Design Phase

Before this phase, the process models may be non-compliant in terms of they do not contain the required controls according the risk assessment of the enterprise. During this phase, a Compliance expert goes through the relevant Business Process model, as it may be delivered by an ERP vendor, step by step. First, the Compliance expert selects an activity contained in the process model. Then he selects a certain control pattern from the control pattern repository. He instantiates the selected pattern by configuring it according to the enterprise’s specific requirements. He then stores the control: a) the control is stored in the SemanticMirror and b) the currently selected activity in the process model is extended by the control (Business Process model adaptation).

5.2 Phase 2 - Recovery Action Design Phase

After a new control is created in the SemanticMirror, the according Business Process expert is notified about this fact. He checks the recovery action part of the control and, if necessary, he modifies/extends the recovery action model of the control. After this phase, the control in the SemanticMirror and the process model in the BP repository are updated with necessary modifications done by the Business Process expert (Business Process model adaptation).

In the following we go into greater details on Business Process model adaptation, which occurs in phase 1 and 2.

Business Process Model Adaptation

A process model is originally in a control-free form. After phase 1 and 2 not only a required control is stored in the SemanticMirror, but also the process model is extended by the required control. The control in the process model makes sure that the process model is executed in a compliant way. Later on, during phase 3, the control in the SemanticMirror monitors that the controls are effective, i.e. that they operate as designed, which is required by law. This way, even if a designed control in the Business Process model during this sub-phase is removed from the Business Process Model by a Business Process expert/Developer who is not aware of compliance requirements, the control in the SemanticMirror will still detect that a control violation occurred.

Below we show the process model adaptation for a process model p by a control c as an example:

Let *p* be the original control-free process model, *c* the required control, the control scope for *c* be *before activity n*, and the recovery action model selected be *Retry & Notify & Recover(r)* with *r* being a pre-designed recovery process for process *p* in case of violation of *c*.

Then the process model adaptation is as follows:

create(p, cd);create(p,cn); transition(p,cd,n,"ok");transition(p,cd,cn,"notOK"); transition(p,cn,before(p,n),"default");⊗ r;

where *cn* is an activity that generates a notification message in case of violation of c; *cd* is an activity-node of type decision; the operator *create(p,a)* creates a node *a* in the process model *p*; the operator *transition(p,a,b,m)* creates a transition in the process model *p* from node *a* to *b* when message *m* is generated after processing node *a;* the operator before(p,a) returns the position of the activity immediately before activity a in the process model *p*; and finally the operator *⊗ r* creates an instance of the process model *r*.

The process model adaptation described above for the selected recovery action model is visualized in Figure 10.

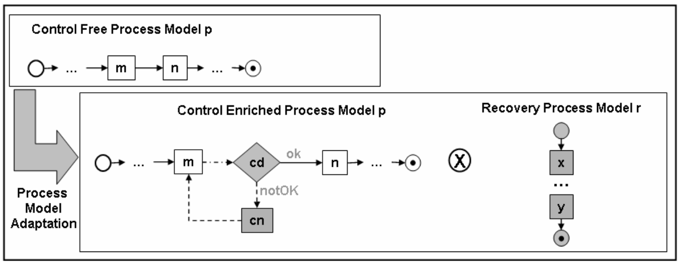


Fig. 10. Process Model adaptation for Retry&Notify/Recover (r)

The cooperative interactions of the actors and the systems during phase 1 and 2 are summarized in Figure 11.

5.3 Phase 3 - Business Process Execution Phase

This phase enables the bidirectional interaction between BPM and Internal Controls management (see Figure 2): The SemanticMirror will be updated by information about the current instance of the Business Process enacted and if a control is violated, the recovery action defined in the control will be executed.

In order to enable the automated generation of the SemanticMirror during execution time, it has to be continuously updated when an activity is performed in the given Business Process instance. The update of the SemanticMirror is done by the introduction of a Knowledge Base of Activities (*KBA*) enacted during execution of a Business Process. With the help of KBA, the current context of the Business Process instance (i.e. all relevant CEs) can be provided to the SemanticMirror.

The *KBA* updating the SemanticMirror are orthogonal to the Business Process management and we introduce them on the conceptual level. The update mechanism

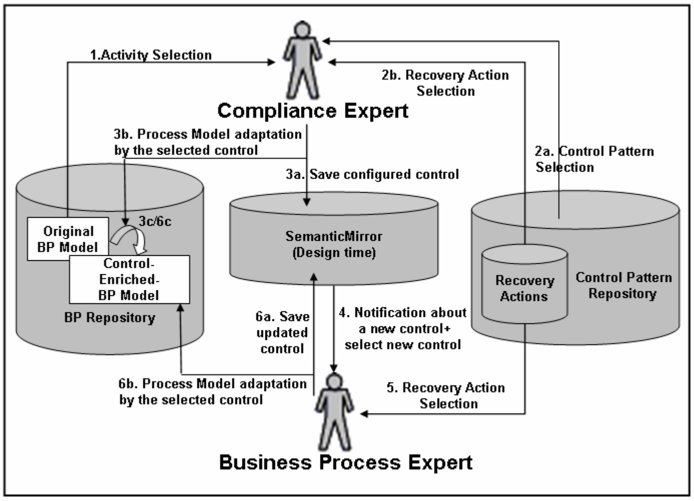

1.Activity Selection
2b. Recovery Action Selection
Compliance Expert
3b. Process Model adaptation by the selected control
2a. Control Pattern Selection
3a. Save configured control
3c/6c
Original BP Model
Control-Enriched-BP Model
BP Repository
SemanticMirror (Design time)
Recovery Actions
Control Pattern Repository
6a. Save updated control
4. Notification about a new control+ select new control
6b. Process Model adaptation by the selected control
5. Recovery Action Selection
Business Process Expert

Fig. 11. Phase 1 and phase 2

of the SemanticMirror is dependent on the underlying BPM infrastructure, i.e. its description is a technical issue. However, in all the cases the destination of a KBA on an implementation level is a network of addressable device such as Trace/log files, a RDBMS, or a messaging destination such as a MQSeries/JMS's Topic/Queue. The destination of the KBAs can be the SemanticMirror itself, when the underlying process execution infrastructure implements the observer design pattern or the command design pattern [6].

In the following we describe the validation of control c during execution time of Business Process p with a recovery action on violation of c *Retry & Notify & Recover(r).* All steps are visualized in Figure 12:

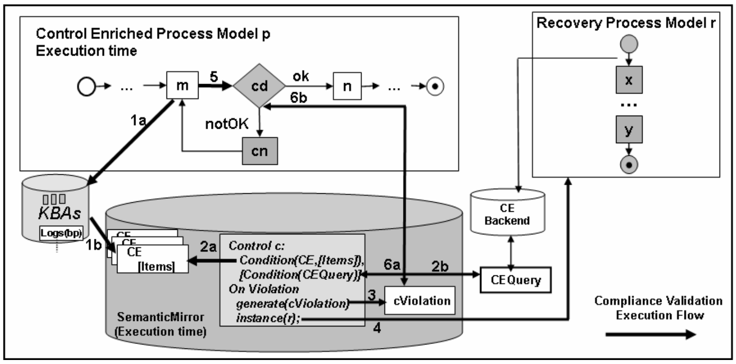


Fig. 12. Phase 3

- 1a. The process context is written to a KBA. Note that this can be done directly on the SemanticMirror itself (updating facts directly in it) depending on the underlying BPM Engine implementation. In this case, go to step 2a.

- 1b. The log entries are extracted and corresponding CE-facts are created and updated in the SemanticMirror.
- 2a. As the state of the SemanticMirror changes in terms of adding/updating CE facts to it, the trigger of control c gets activated. The condition of c is determined by the values of the CE facts in the SemanticMirror itself or
- 2b. optionally by querying the necessary backend systems using the CEQuery of a subjected CE.
- 3. If the conditions of the controls are violated, a new fact in the SemanticMirror (*cViolation*) will be generated signaling that control c has been violated.
- 4. An instance of the recovery process r is generated.
- 5. The instance p steps into the decision node cd.
- 6a. cd, being a decision node, is a coordinator activity. The activity of cd queries the SemanticMirror for a fact instance called *cViolation*.
- 6b. In case of existence of a *cViolation* in the SemanticMirror, cd sets the transition to "*ok*", otherwise to "*notOK*".

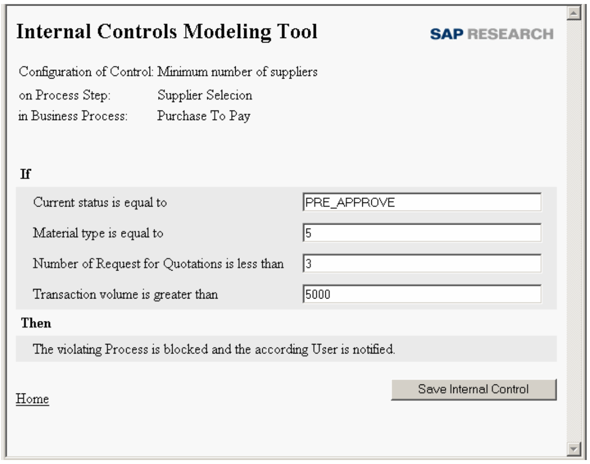
Please notice that the approach described above will still detect a control violation in the SemanticMirror, even if a Business Process expert/technical consultant will remove the control from the process model being not aware of the necessity of that control: the process context is always written to the SemanticMirror during step 1a/1b and the controls exist independently in the SemanticMirror. Further, the described approach enables dynamical application of the controls during the execution phase of a Business Process. There is a minimum overlap between Business Process design and compliance design. Thus, new application controls can be designed for Business Processes by adding new control statements to the SemanticMirror while the original design of the Business Process requires no manual change, which is one of the main advantages of our approach.

6 Implementation

Besides the conceptual soundness, one of the challenges in such a kind of approaches is their efficient and scalable implementation. There are two open issues that have to be discussed from the implementation point of view: 1) How to design and execute the Business Processes and 2) How to implement the SemanticMirror.

Regarding the first issue, we have selected to implement a prototype based on JBoss jBPM (JBoss, http://www.jboss.com). We have decided to use jBPM since it offers concepts such as task management and identity management, which allowed us to completely simulate typical ERP scenarios necessary for our experimental environment. The basis for the implementation of **SemanticMirror** is the model of the Internal Controls (see section 3). We have decided to implement the controls as Event-Condition-Action (ECA) rules. The Dwyer patterns and scopes [5] can be mapped to ECA rules (as already shown in [12]), thus the control patterns and scopes can be mapped to them. We use the JBoss Rule Engine (also known as Drools) implementing the RETE-Algorithm.

For the task of updating the SemanticMirror during execution time of Business Processes (Phase 3 of the approach), we use facilities provided by jBPM Engine implementing the command software design pattern [6]: jBPM provides the possibility to register (during design-time) a so called *ActionHandler* to each node-class (activity) of a Process definition (called jPDL in jBPM) with additional custom functionality. Our implementation of the *ActionHandler*-Interface (*SemanticMirrorSynchronizer)* obtains a reference to the SemanticMirror (in terms of obtaining a reference to Rule Engines working memory) and the current instance of the *process context* provided automatically by the jBPM Process Engine to *SemanticMirrorSynchronizer* is added to the SemanticMirror. Additionally each Decision-Node in jPDL can be equipped with a *DecisionHandler,* which determines which transition to take. Our implementation of the DecisionHandler establishes a connection to the SemanticMirror and queries it for determining an instance of the *cViolation*-fact.



Configuration of Control:	Minimum number of suppliers
on Process Step:	Supplier Selecion
in Business Process:	Purchase To Pay

If

Current status is equal to	PRE_APPROVE
Material type is equal to	5
Number of Request for Quotations is less than	3
Transaction volume is greater than	5000

Then

The violating Process is blocked and the according User is notified.

Fig. 13. A snapshot of the Internal Controls Modeling Tool

Further, we are currently in the process of implementing an Internal Controls Modeling tool based on the high-level control pattern repository for the Compliance experts. The java-based implementations of CEs (activities, business documents, users, and controls) are equipped with additional information specifying to which control patterns in which Business Process models they may occur using the java annotation mechanism. In this way, after a control pattern is instantiated via the so called Internal Controls Modeling Tool based on a concrete control for a specific Business Process, all the necessary subjected and objected CEs for designing/configuring that control are automatically proposed to the Compliance expert. Figure 13 shows a snapshot of the Internal Controls Modeling Tool for an already instantiated control “Minimum number of suppliers” which satisfies the “Limit checks” control pattern.

7 Related Work

On a conceptual level our work is related to [4], where a taxonomy of risks in Business Processes is provided. It does not explicitly state how a risk is positioned inside the Internal Controls compliance domain and leaves the semantic link between risks, Business Process design and execution open. [8] presents a logical language, called PENELOPE, that provides the ability to verify temporal constraints arising from compliance requirements on effected Business Processes. The contributions of this paper are: i) providing an overall methodology for a model driven approach to Business Process compliance; ii) offering a technique for process model enrichment based on a finite extensible set of control patterns; and iii) proposing an explicit realization of guarding the compliant behaviour of a Business Process during its execution.

Significant research exists on the modeling of control flow in Business Processes by usage of patterns to identify commonly used constructs [www.workflowpatterns.com]. On a similar note, [13] provides temporal rule patterns for regulatory policies, although the objective of this work is to facilitate event monitoring rather than the usage of the patterns for support of compliance in Business Processes. Further a conceptual model based on UML Profile is defined there as a basis for defining compliance rules. But the work does not explicitly state how to reason over the UML Profiles and how they may be related to the Business Process execution level of enterprises.

Casati et al have contributed significant work in [14] for pattern-based exception handling in workflows, which we consider as highly related to our pattern-based approach. Especially the proposed algorithms for pattern instantiation and specialization can be reused for application to the high level control patterns proposed in our work.

On a technical level the work done by Robinson in [12] we consider as related to ours. He uses monitoring patterns based on the Dwyer patterns [5] expressed as rules (in an extension of Jess) to monitor the runtime behavior of a system to verify whether runtime execution satisfies its design time specification. The difference is that we are concerned with the separation of Business and Internal Controls process in order to keep entities developed in both areas better adaptable and reusable, where his work is located in the area of system requirement engineering, in particular system verification.

8 Conclusion

In this paper we introduced a pattern based approach for modeling Internal Controls required by regulations such as SOX. They can be captured as declarative rules and checked during execution-time on Business Processes. We built the model based on the de facto Internal Controls standard called COSO. The approach supports the definition of the controls outside of the workflow in order to enable the reuse of process models and controls in different business environments.

Currently our approach requires the manual selection of a concrete control pattern and its specific design on a Business Process according to the enterprise-specific compliance needs. A higher level of automation can be brought to the approach by building a "Risk Repository" as a starting point of the approach.

Another issue that must be addressed is the inter-control dependency: in order to become effective, a "well-designed" control may depend on existence, effective design, and operation of other controls. This issue is also mentioned directly by law [9]. We currently recognize this fact by introducing the "related to" attribute in a pattern specification. On a similar note, different designed controls can contradict, subsume or block each other in a Business Process. We have to extend the Control Design phase by concepts to detect such situations.

References

1. Pub. L. 107-204. 116 Stat. 754, Sarbanes Oxley Act (2002)
2. Committee of Sponsoring Organizations of the Treadway Commission (COSO), Internal Control - Integrated Framework (1992)
3. Hartman, T.: The Cost of Being Public in the Era of Sarbanes-Oxley (June 2005)
4. zur Muehlen, M., Rosemann, M.: Integrating Risks in Business Process Models. In: ACIS 2005. Proceedings of the 2005 Australasian Conference on Information Systems, Manly, Sydney, Australia, November 30-December 2 (2005)
5. Dwyer, M., Avrunin, G., Corbett, J.: Patterns in Property Specification for Finite-State Verification. In: Proceedings of the 21st International Conference on Software Engineering, pp. 411–420 (May 1999)
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Element of Reusable Object Oriented Software. Addison-Wesley, Reading (1995)
7. Davenport, T., Short, J.: The New Industrial Engineering: Information Technology and Business Process Redesign. Sloan Management Review 31, 11–27 (1990)
8. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes from Obligations and Permissions. In: BPD 2006. 2nd Workshop on Business Processes Design Proceedings (2006)
9. Public Company Accounting Oversight Board (PCAOB), PCAOB Accounting Standard No. 2, Paragraph 12
10. Namiri, K., Stojanovic, N.: A Formal Approach for Internal Controls Compliance in Business Processes. In: BPMDS 2007. 8th Workshop on Business Process Modeling, Development, and Support conjunction with CAiSE 2007 (2007)
11. Sadiq, S., Governatori, G., Kioumars, N.: Modeling Control Objectives for Business Processes. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, Springer, Heidelberg (2007)
12. Robinson, W.: Implementing Rule-based Monitors within a Framework for Continuous Requirements Monitoring, HICSS 2005, Hawaii, USA (2005)
13. Giblin, C., Muller, S., Pfitzmann, B.: From regulatory policies to event monitoring rules: Towards model driven compliance automation. IBM Research Report. Zurich Research Laboratory (October 2006)
14. Casati, F., Castano, S., Fugini, M., Mirbel, I., Pernici, B.: Using Patterns to Design Rules in Workflows. IEEE Transactions on Software Engineering 26(8) (August 2000)

Constraint-Based Workflow Models: Change Made Easy

M. Pesic[1], M.H. Schonenberg[2], N. Sidorova[2], and W.M.P. van der Aalst[2]

[1] Department of Technology Management
[2] Department of Mathematics and Computer Science,
Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{m.pesic, m.h.schonenberg, n.sidorova, w.m.p.v.d.aalst}@tue.nl

Abstract. The degree of flexibility of workflow management systems heavily influences the way business processes are executed. Constraint-based models are considered to be more flexible than traditional models because of their semantics: everything that does not violate constraints is allowed. Although constraint-based models are flexible, changes to process definitions might be needed to comply with evolving business domains and exceptional situations. Flexibility can be increased by run-time support for dynamic changes – transferring instances to a new model – and ad-hoc changes – changing the process definition for one instance. In this paper we propose a general framework for a constraint-based process modeling language and its implementation. Our approach supports both ad-hoc and dynamic change, and the transfer of instances can be done easier than in traditional approaches.

1 Introduction

When supporting business processes there is a difficult trade-off to be made. On the one hand, there is a desire to control processes and to avoid incorrect or undesirable executions of these processes. On the other hand, workers want flexible processes that do not constrain them in their actions. This apparent paradox has limited the application of workflow management systems thus far since, as indicated by many authors, workflow management systems are too restrictive and have problems concerning dealing with change [3].

Many approaches to resolve the paradox have been proposed. Some of them try to avoid change, e.g. by generating implicit alternative paths [6,8], or by differing the selection of the desired behavior [7]. Others allow for changing the model for a single instance and/or changing a process model while migrating all instances [9,11,19,23]. The migration of process instances from one model to another introduces many interesting problems [3,9,19,23]. For example, the "dynamic change bug" originally described in [11] shows that it may be impossible to put the process instance into a suitable state of the new model without skipping or repeatedly executing tasks.

In this paper we propose a solution that has some features of both approaches: we try to avoid the need for change and at the same time we provide full support for all kinds of change. To avoid the need for change we address the following problem: Traditional workflow languages force or stimulate the designer to *over-specify* things. For example, it is possible to model all kinds of choices in today's systems. It is, however, not possible to simply state that two activities should never occur together. Instead, the user is forced to provide a detailed strategy to implement this simple requirement, e.g. by introducing a decision task and deciding when and by whom this task is executed. We believe that replacing the imperative approach with a *declarative* one is essential for making workflow management more flexible. Therefore, we consider here a framework where workflows are defined by constraint models.

Avoiding over-specification makes processes more flexible (more execution paths are allowed) and allows avoiding costly changes. Change is sometimes still unavoidable because of exceptions (e.g., an important customer has a special request which requires the violation of a business rule) or changed circumstances (e.g., a new law enforcing to reverse the order of two activities). This paper explores *what change means in the context of constraint-based languages*. Here it is interesting to see whether similar problems as reported in [3,11,19] occur. Surprisingly, it turns out that both ad-hoc and evolutionary changes are rather easy to support. This explains the subtitle of this paper: "Change Made Easy".

The results we report here show that it is possible both (1) to avoid the need for unnecessary changes and restrictions (using a more declarative style) and (2) to provide support for changes at the instance level (ad-hoc change) and at the type level (evolutionary change). Moreover, it is also possible to easily differentiate between mandatory and optional constraints. A user is forbidden to violate a mandatory constraint or to change a model so that a mandatory constraint becomes violated. For optional constraints a warning is generated and the user may choose to violate it or not. Note that in each case model checking techniques can give good diagnostic information that helps the user to understand potential problems.

Our framework is supported by the *ConDec* language [16]. ConDec is a graphical declarative process modeling language supported by the *Declare* tool, which also supports related languages such as *DecSerFlow* [4]. The Declare workflow management system is open in the sense that it can support multiple constraint-based languages and each of the languages is extendible and can be changed without changing the engine. This is achieved by a flexible mechanism mapping graphical constraints onto LTL (Linear Temporal Logic) [14]. Note that the semantics are expressed in a temporal logic but the end user only sees the graphical notation when modeling. The Declare system fully supports the approach presented in this paper and the software can be downloaded from `http://is.tm.tue.nl/staff/mpesic/declare.htm`.

The remainder of the paper is organized as follows. The general framework of constraint modeling and changes are presented in Section 2. In Section 3 we describe ConDec, an implementation of a constraint modeling language, based on the general framework. ConDec is supported by the Declare tool. The support of change in ConDec is described in Section 4. Section 5 discusses related work, and finally Section 6 concludes the paper and gives directions for future work.

2 Constraint Models

Constraint models are suitable for supporting flexible processes that allow many different executions. Most theoretical process modeling languages, such as Petri Nets [17], process algebras [15] and more applied business languages like BPMN, UML and EPCs [20] define direct causal relationships between activities in process models. Opposed to this, constraint-based languages are of a less procedural nature and use a more declarative style. Using constraints, the behavior is restricted. Unlike procedural languages constraints may be non-local, e.g., *"eventually A is followed by B"* and negative, e.g., *"either A or B can occur but not both"*.

Activities and constraints on activities are the key elements of a constraint-based model. We distinguish the universe of all activities $\mathcal{A}$ and the universe of all constraints $\mathcal{C}$. $\mathcal{A}^*$ denotes the set of all sequences over $\mathcal{A}$. We say that c is a constraint over $A \subseteq \mathcal{A}$, if it does not mention any activity $a \notin A$. A constraint is a boolean expression that evaluates to true or false for every trace $\sigma \in \mathcal{A}^*$. If a constraint c evaluates to true for a trace $\sigma \in \mathcal{A}^*$, then we say that σ satisfies c, denoted as $\sigma \models c$, otherwise we denote it as $\sigma \not\models c$. In Section 3 we will show that such constraints can be expressed graphically and be mapped onto LTL.

Example 1 (Constraint). Let $A = \{curse, pray\}$ be a set of two activities and $c =$ *"Every curse activity is eventually followed by a pray activity"* be a constraint over A. Then $\langle curse\rangle \not\models c$, $\langle pray, pray\rangle \models c$, $\langle curse, curse, pray\rangle \models c$ and $\langle curse, pray, curse\rangle \not\models c$.

2.1 Preliminaries

First we introduce sequence concatenation ($\frown$), function overriding ($\oplus$) and reduction ($\ominus$), which are used in the remainder of the framework. Sequences can be concatenated into a new sequence.

Definition 1 (Sequence concatenation $\frown$). *Let σ, γ be sequences over $\mathcal{A}$ with $\sigma = a_1, a_2, a_3, \ldots, a_n (\forall a_i \in \mathcal{A})$ and $\gamma = b_1, b_2, b_3, \ldots, b_n (\forall b_i \in \mathcal{A})$. Then $\sigma \frown \gamma = a_1, a_2, a_3, \ldots, a_n, b_1, b_2, b_3, \ldots, b_n$.*

We use function overriding to add and remap elements of a function domain and we use function reduction to remove elements from a function domain. When a function f is undefined for an element a, we denote this as $f(a) = \perp$.

Definition 2 (Function overriding $\oplus$). *Let $f : A \rightarrow B$. Then $f \oplus (a,b) : A \cup \{a\} \rightarrow B \cup \{b\}$ such that $(f \oplus (a,b))(a) = b$ and $\forall x \in A \backslash \{a\} : (f \oplus (a,b))(x) = f(x)$.*

Definition 3 (Function reduction $\ominus$). *Let $f : A \rightarrow B$, $a \in A$, $b \in B$. Then $f \ominus (a,b) : A \backslash \{a\} \rightarrow B \backslash \{b\}$ such that $(f \ominus (a,b))(a) = \perp$ and $\forall x \in A \backslash \{a\} : (f \ominus (a,b))(x) = f(x)$.*

2.2 Constraint Workflows

Constraints define the boundaries within which activities can be executed. Besides activities and constraints on these activities, the constraint model also includes a mapping that defines whether constraints are optional (may be violated) or mandatory (may never be violated).

Definition 4 (Constraint Model cm). *A constraint model cm is defined as a triple $cm = (A, C, c_{type})$, where*

- *$A \subseteq \mathcal{A}$ is a set of activities in the model;*
- *$C \subseteq \mathcal{C}$ is a set of constraints where every element $c \in C$ is a constraint over A;*
- *$c_{type} : C \rightarrow \{mandatory, optional\}$ is a function that defines whether constraints are mandatory or optional.*

We use $\mathcal{U}_{CM}$ to denote the universe of all constraint models.

For convenience, we define an operation to remove optional constraints from a constraint model.

Definition 5 (Mandatory version of *cm***).** *Let $cm = (A, C, c_{type})$ be a constraint model, then $mand(A, C, c_{type}) = (A, C', c'_{type})$, where $C' = \{c \in C \mid c_{type}(c) = mandatory\}$ and $c'_{type} : C' \rightarrow$*
$\{mandatory\}$.

A constraint workflow contains several running instances, each related to a constraint model and a sequence of actions performed by the instance up to the current moment. The framework we develop here, should support changes of the constraint model by redefining restrictions on the behavior at run time. Moreover, we want to be able to change a constraint model for a cluster of instances, which could be e.g., all instances related to the handling of complaints at an insurance department. For this purpose, we introduce the notion of constraint model identifiers. This identifier is then mapped to one of the constraint models from the universe. Instances, in their turn, are mapped to a constraint model identifier.

Definition 6 (Constraint Workflow *wf***).** *A workflow specification based on constraint models is defined by the tuple $wf = (P_{id}, cm_{id}, Pmap, CMmap, trace)$, where*

- *P_{id} is a set of process identifiers (instances);*
- *cm_{id} is a set of constraint model identifiers;*
- *$Pmap : P_{id} \rightarrow cm_{id}$ is a function that maps instances to model identifiers;*
- *$CMmap : cm_{id} \rightarrow \mathcal{U}_{CM}$ is a function that maps model identifiers to constraint models;*
- *$trace : P_{id} \rightarrow \mathcal{A}^*$ is a function that maps instances to execution traces.*

We use $\mathcal{U}_{WF}$ to denote the universe of all constraint workflows wf.

Figure 1 depicts a mapping from instances of P_{id} to model identifiers in cm_{id} and a mapping from these model identifiers to constraint models in $\mathcal{U}_{CM}$. $CMmap(cm_{id})$ results in a constraint model cm. Note that not all constraint models in the universe $\mathcal{U}_{CM}$ need to

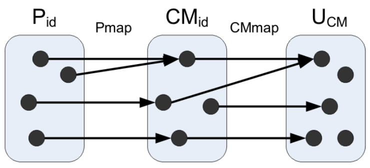


Fig. 1. Mappings

have a related model identifier in cm_{id}. Also observe that different process instances can be mapped onto the same constraint model identifier, i.e., the same constraint model. Even different constraint model identifiers might be mapped onto the same constraint model, which reflects situations in which the same constraint model is used in different contexts (e.g., two companies can develop two identical models).

Satisfaction of constraint sets depends on the execution trace.

Definition 7 (Satisfaction of constraint sets). *Let C be a set of constraints and $\sigma \in \mathcal{A}^*$ then $\sigma \models C \iff \forall c \in C : \sigma \models c$ and $\sigma \not\models C \iff \exists c \in C : \sigma \not\models c$.*

The purpose of constraints is, however, to define conditions that should hold on the completed traces of instances. During execution we can only evaluate prefixes of those traces, and a constraint violation on a prefix does not necessarily imply that the constraint will be violated on the completed trace. Therefore, we introduce an evaluation function that determines whether a constraint model *cm* is *satisfied*, *violated* or *temporarily violated*, i.e., although the trace currently violates the constraints, the constraints can still become satisfied in the future.

Definition 8 (Evaluation *eval***).** *Let $cm = \in \mathcal{U}_{CM}$ be a constraint model where $cm = (A, C, c_{type})$ and $\sigma \in \mathcal{A}^*$ be a trace. Then the evaluation function eval is defined as*

$$eval(\sigma, (A, C, c_{type})) = \begin{cases} \textit{satisfied} & \textit{if } \sigma \models C; \\ \textit{temporarily violated} & \textit{if } (\sigma \not\models C) \wedge (\exists \gamma \in A^* : \sigma^\frown\gamma \models C); \\ \textit{violated} & \textit{otherwise.} \end{cases}$$

Example 2. (Satisfaction) Consider again constraint c, with $c =$ *"Every curse activity is eventually followed by a pray activity"*. Suppose trace $\sigma = \langle curse, curse \rangle$ at some moment during execution. Obviously $\sigma \not\models c$, but σ can be a prefix of a trace that could satisfy c during execution. For example $\langle curse, curse, pray \rangle \models c$.

When an instance executes actions, the trace of that instance is updated.

Definition 9 (Execution *exec***).** *Let $wf \in \mathcal{U}_{WF}$ be a constraint workflow where $wf = (P_{id}, cm_{id}, Pmap, CMmap, trace)$. Let $p_{id} \in P_{id}$, $CMmap(Pmap(p_{id})) = cm$, $cm = (A, C, c_{type})$ and $a \in A$. Then the execution function exec is defined as*

$$exec(wf, a, p_{id}) = (P_{id}, cm_{id}, Pmap, CMmap, trace \oplus (p_{id}, trace(p_{id})^\frown \langle a \rangle)).$$

Furthermore we call an execution:

- ***normal*** *if no constraint is violated: $eval(trace(p_{id})^\frown\langle a \rangle, cm) \neq violated$;*
- ***deviating*** *if only optional constraints are violated: $eval(trace(p_{id})^\frown\langle a \rangle, cm) = violated \wedge eval(trace(p_{id})^\frown\langle a \rangle, mand(cm)) \neq violated$;*
- ***invalid*** *otherwise.*

If an instance is closed, the instance will be removed, together with its trace. Unlike in procedural language where an instance is closed automatically when some closing state is reached, instances of constraint-based models can have multiple states at which the instance could be closed. Therefore, many strategies can be used to close an instance of a constraint-based model. One example of a closing strategy would be to allow users to explicitly choose when to close an instance and to allow only "normal" closings.

Definition 10 (Closing an instance *close***).** *Let $wf = \in \mathcal{U}_{WF}$ be a constraint workflow where $wf = (P_{id}, cm_{id}, Pmap, CMmap, trace)$. Let $p_{id} \in P_{id}$, $Pmap(p_{id}) = cm_{id}$ and $CMmap(cm_{id}) = cm$. Then closing instance close is defined as*

$$close(wf, p_{id}) = (P_{id}', cm_{id}, Pmap', CMmap, trace'),$$

where
$P_{id}' = P_{id} \setminus \{p_{id}\}$, $Pmap' = Pmap \ominus (p_{id}, cm_{id})$ and $trace' = trace \ominus (p_{id}, trace(p_{id}))$. We call closing of an instance:

- ***normal** if all constraints are satisfied: $eval(trace(p_{id}), cm) = satisfied$;*
- ***deviating** if all mandatory constraints, but not all optional constraints are satisfied: $eval(trace(p_{id}), cm) \neq satisfied \wedge eval(trace(p_{id}), mand(cm)) = satisfied$;*
- ***invalid** otherwise.*

Operations. We can easily add an instance to a workflow by extending the instance set and adding an empty trace for this instance to the trace mapping.

Definition 11 (Adding a process instance *add_{PI}***).** *Let $wf \in \mathcal{U}_{WF}$ be a constraint workflow where $wf = (P_{id}, cm_{id}, Pmap, CMmap, trace)$. Let $p_{id} \notin P_{id}$, $cm_{id} \in cm_{id}$, then*

$$add_{PI}(wf, p_{id}, cm_{id}) = (P_{id} \cup \{p_{id}\}, cm_{id}, Pmap \oplus (p_{id}, cm_{id}), CMmap, trace \oplus (p_{id}, \langle\rangle)).$$

Constraint models from the universe can be added to the workflow by adding a constraint model identifier to the identifier set and linking the identifier to the required constraint model.

Definition 12 (Adding a constraint model *add_{CMI}***).** *Let $wf \in \mathcal{U}_{WF}$ be a constraint workflow where $wf = (P_{id}, cm_{id}, Pmap, CMmap, trace)$. Let $cm_{id} \notin cm_{id}$, $cm \in \mathcal{U}_{CM}$, then*

$$add_{CMI}(wf, cm_{id}, cm) = (P_{id}, cm_{id} \cup \{cm_{id}\}, Pmap, CMmap \oplus (cm_{id}, cm), trace).$$

Verification. Once a constraint model has been defined, we can verify whether it contains dead activities and conflicts. We call an activity a dead activity, when all traces that contain this activity violate the constraints. We say that the model contains a conflict when there are no traces that could (eventually) satisfy the constraints.

Definition 13 (Dead activity). *Let $cm = (A, C, c_{type})$ and $a \in A$. Then a is a dead activity if $\forall \sigma \in \mathcal{A}^* : a \in \sigma \Rightarrow \sigma \nvDash C$.*

Definition 14 (Conflict). *Let $cm = (A, C, c_{type})$, then there is a conflict in cm if $\forall \sigma \in \mathcal{A}^* : \sigma \nvDash C$.*

Note that if there is a conflict in the model, then all activities of the model are dead activities.

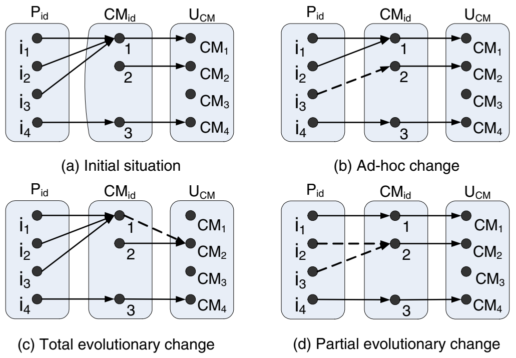


Fig. 2. Changes for constraint workflows

2.3 Change

Constraint models support both ad-hoc and evolutionary changes. Ad-hoc changes are typically needed to handle an exceptional situation for one case. An ad-hoc change for an instance is allowed, when the instance trace satisfies the new model. Evolutionary changes occur when there is a change in the process itself, e.g., by new laws or business strategies. Traces of all running instances of the corresponding process model should be evaluated and, if possible, the instances should be transferred (remapped) to the new model. When this is not possible, i.e., the trace violates the new constraints, we call this a *history violation*.

Figure 2 depicts a constraint workflow (without the *trace* mapping) and is used to illustrate all change types. In Figure 2(a) four instances are depicted, of which i_1, i_2, i_3 are instances of constraint model cm_1. Instance i_4 is an instance of cm_4. Constraint model cm_2 has no instances yet and cm_3 is not part of the constraint workflow. In the remainder of this section we will explain all change types, by describing changes performed on the original workflow, depicted in Figure 2(a). Dashed lines denote a remapping of *Pmap* or *CMmap* with respect to the original workflow.

Definition 15 (Ad-hoc change δ_{AH}). *Let $wf \in \mathcal{U}_{WF}$ be a constraint workflow where $wf = (P_{id}, cm_{id}, Pmap, CMmap, trace)$. Let $p_{id} \in P_{id}$, $cm_{id} \in cm_{id}$* [1], *$eval(trace(pid), CMmap(cm_{id})) \neq violated$, then*

$$\delta_{AH}(wf, p_{id}, cm_{id}) = (P_{id}, cm_{id}, Pmap \oplus (p_{id}, cm_{id}), CMmap, trace).$$

Figure 2(b) shows a possible ad-hoc change. Suppose we would like to perform an ad-hoc change on instance i_3. We want this instance to use constraint model cm_2 instead of cm_1. If the trace of instance i_3 does not violate the constraints of the new model cm_2, we can remap instance i_3 to identifier 2, which maps onto model cm_2.

Total evolutionary changes can only be performed when all instances satisfy the new model. If this is the case, all instances will be transfered to the new model.

[1] Note that if $cm_{id} \notin cm_{id}$, operation add_{CMI} could be executed first.

Definition 16 (Total evolutionary change $\delta_{E_{total}}$). *Let $wf \in \mathcal{U}_{WF}$ be a constraint workflow where $wf = (P_{id}, cm_{id}, Pmap, CMmap, trace)$. Let $cm_{id} \in cm_{id}$, $cm \in \mathcal{U}_{CM}$, $\forall p_{id} \in Pmap^{-1}(cm_{id}) : eval(trace(p_{id}), CMmap(cm_{id})) \neq violated$, then*

$$\delta_{E_{total}}(wf, cm_{id}, cm) = (P_{id}, cm_{id}, Pmap, CMmap \oplus (cm_{id}, cm), trace).$$

In Figure 2(c) we illustrate an example of a total evolutionary change. Suppose we would like to transfer all instances of constraint model cm_1 (instances mapped to identifier 1) to cm_2. When for all instances the current trace satisfies cm_2, we can remap identifier 1 to cm_2.

We also define partial evolutionary change, in which only instances that satisfy the new model are transferred to the new model. All other instances proceed their execution according to the old model.

Definition 17 (Partial evolutionary change $\delta_{E_{partial}}$). *Let $wf \in \mathcal{U}_{WF}$ be a constraint workflow where $wf = (P_{id}, cm_{id}, Pmap, CMmap, trace)$. Let $cm_1 \in cm_{id}$, $cm_2 \in cm_{id}$, then*

$$\delta_{E_{partial}}(wf, cm_1, cm_2) = (P_{id}, cm_{id}, Pmap', CMmap, trace),$$

where $Pmap' : P_{id} \rightarrow cm_{id}$, such that

$$Pmap'(p_{id}) = \begin{cases} cm_2 & , \forall p_{id} \in SatP_{id} \\ Pmap(p_{id}) & , \forall p_{id} \in P_{id} \setminus SatP_{id}. \end{cases}$$

and
$SatP_{id} = \big\{ p_{id} \in P_{id} \mid Pmap(p_{id}) = cm_1 \wedge eval\big(trace(p_{id}), CMmap(cm_2)\big) \neq violated \big\}.$

An example of partial evolutionary change is given in Figure 2(d). Again, suppose we would like to transfer instances of constraint model cm_1 (instances mapped to identifier 1) to cm_2. Then all instances that satisfy cm_2 are remapped to an identifier that is related to cm_2. Note that for instance i_1 it is not possible to migrate. Therefore, it remains an instance of cm_1.

Change in imperative models is hindered by the fact that an equivalent new state must be found in the new model, which is not always possible [11]. For declarative models it is straightforward to transfer instances. Instances for which the current trace satisfies the constraints of the new model, are mapped onto the new model. Hence the "dynamic change bug" described in [11] does not apply. In the next section we will present an implementation based on this framework.

3 ConDec and Declare

In this section we briefly introduce ConDec [16], a constraint-based process modeling language, based on the framework we presented in Section 2. ConDec is supported by the Declare tool, see Section 3.2.

3.1 ConDec

ConDec uses *an open set of constraint templates* for the definition of relationships between activities. Each template has (1) a name, (2) a graphical representation and (3)

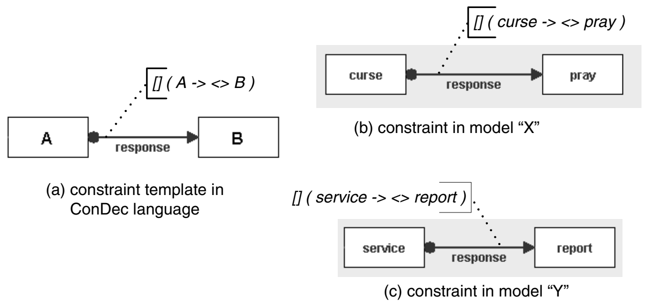

[] (A -> <> B)
A
response
B
(a) constraint template in ConDec language
[] (curse -> <> pray)
curse
response
pray
(b) constraint in model "X"
[] (service -> <> report)
service
response
report
(c) constraint in model "Y"

Fig. 3. Constraint template "response" and two "response" constraints

semantics given by a Linear Temporal Logic (LTL) formula on finite traces [13]. LTL is a temporal logic that, in addition to classical logical operators, uses several temporal operators: always ($\Box$), eventually ($\Diamond$), until ($\sqcup$) and next time ($\bigcirc$) [14]. LTL formulas can be added to the language by means of constraint templates. Constraint templates are parameterized graphical representations of LTL formulas. Templates can easily be added, removed and changed in ConDec.

Figure 3 shows a constraint template and its application to two models. For the sake of clarity, we have also added the corresponding LTL formulas to the picture. The template depicted in Figure 3(a) is the response template and it is defined as a single line with special symbols between some activities "A" and "B", i.e., "A" and "B" are parameters of the template. The semantics of the template are given by the formula $\Box(A \rightarrow \Diamond B)$: every execution of activity "A" should eventually be followed by at least one execution of activity "B". The response template can be used to create response constraints in various ConDec process models, by replacing template parameters with activities from the model. Figures 3(b) and 3(c) show parts of two ConDec models, each containing a response constraint between two activities.

Defining templates in this way enables adding various types of relations between activities in ConDec. More than twenty LTL-based constraint templates are described in [5]. The great benefit of constraint templates is that LTL formulas are hidden from the users, therefore they do not have to be LTL experts in order to understand underlying formulas.

Process Modeling in ConDec. ConDec models are suitable for supporting flexible processes with many deviations during the execution. As an example, consider the process for a car rental shop. The model of the car rental process is given in Figure 4. Initially, the client gets registered (activity "register client data"). The client will be charged (activity "charge") for the rental and (if applicable) all damage he caused. The "charge" activity will occur at least once, but the moment of charging is not fixed. If during the rental period a problem is identified (activity "identify problem"), then car will be checked (activity "check"). During the rental period the client can request repairs (activity "request") on which the car rental shop will repair the car if necessary (activity "service") and include the findings in the maintenance report of the car (activity "report") at a suitable moment. The client could request many repairs, or none at all and it is not known in advance when requests will be made.

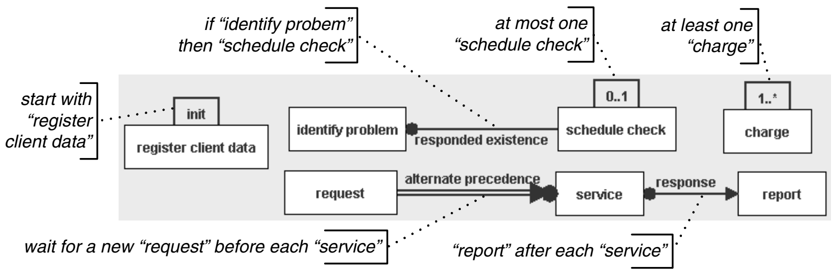
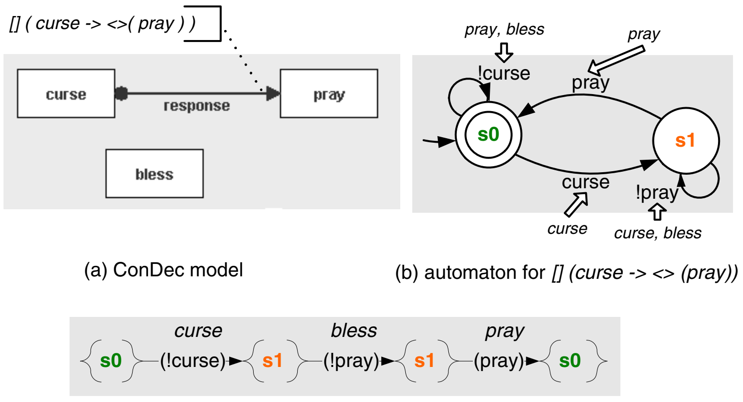

if "identify probem" then "schedule check"
at most one "schedule check"
at least one "charge"
start with "register client data"
init
register client data
identify problem
responded existence
0..1
schedule check
1..*
charge
request
alternate precedence
service
response
report
wait for a new "request" before each "service"
"report" after each "service"

Fig. 4. Activities and constraints in car rental example

To model the the process of the car rental shop, we add several constraints on the execution of the activities in the shop. Every instance must start with registering the client (constraint "init"). The client will be charged for the rental and for all caused damage, so he will be charged at least once (constraint "1..*"). Every repair service on the car will only be done on request of the client, i.e., there has to be at least one occurrence of activity "request" between each two occurrences of activity "service". Note that other activities may be executed in between "request" and "service". Also, for every service, eventually a report must be generated (constraint "response"). The car must be checked when a problem is identified (constraint "responded existence"). However, in case of a car with a long period without checks, employees can decide to schedule check even if no serious problems were identified. At most one check will be performed during rental (constraint "0..1").

Process Execution. Execution of activities in a process instance creates a history trace for that instance (cf. Definition 9). The history of a process instance is a chronologically ordered list of events that occurred in the instance. During execution, the state of every constraint (cf. Definition 8) is depicted by a color: (1) green for satisfied, (2) orange for temporarily violated and (3) red for violated. We do not allow the execution of activities that would permanently violate mandatory constraints (depicted by solid lines). The user will be warned for violation of optional constraints (depicted by dashed lines), but he is free to choose to violate optional constraints. Closing an instance is only allowed when all mandatory constraints are satisfied. Again, warnings are given when an instance that is closed does not satisfy all optional constraints, but the user is free to close the instance anyway.

Constraint semantics (expressed in LTL formulas) are used for the automated execution of ConDec models. Every constraint (LTL formula) is translated into a finite automaton [13]. The constraint is satisfied when the automaton is in an accepting state. If the automaton is not in an accepting state and an accepting state is still reachable, the constraint is temporarily violated. The constraint is permanently violated when the automaton is not in an accepting state and an accepting state is not reachable. Also, one overall automaton (*mandatory automaton*) is generated for the conjunction of LTL formulas of all *mandatory* constraints in the model, and it is used to decide which activities can be executed without violating mandatory constraints (cf. Definition 9).



(a) ConDec model (b) automaton for [] (curse -> <> (pray))

(c) history: curse, bless, pray

Fig. 5. Illustrative example - an instance with history ⟨curse, bless, pray⟩

For illustration purposes we use a simple ConDec model with three activities (“curse”, “pray” and “bless”) and only one constraint, as shown in Figure 5(a). The response constraint specifies that after every execution of the activity “curse” at least one execution of the activity “pray” has to follow (i.e., $\Box(curse \rightarrow \Diamond(pray))$).

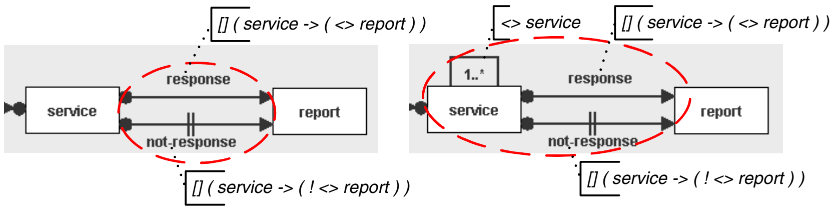
The automaton corresponding to the constraint in the ConDec model is shown in Figure 5(b). The automaton has two states $\{s0, s1\}$. The initial state is `s0`, which is also the accepting state of the automaton[2]. Executing an activity triggers a transition of the automaton.

Let us assume that an instance of the model presented in Figure 5(a) has history ⟨curse, bless, pray⟩. Figure 5(c) shows how this execution history determines the states of the automaton [3]. Initially, the automaton is in its accepting state `s0` (the response constraint is satisfied). Next, activity “curse” triggers a transition to state `s1`. State `s1` is not an accepting state, but an accepting state (`s0`) is reachable from it (the response constraint is temporary violated). Execution of activity “bless” triggers transition “`!pray`” and the automaton remains in state `s1`. Finally, activity “pray” transfers the automaton to accepting state `s0` (the constraint model is satisfied again).

Verification of ConDec models. For correct execution it is important that models do not contain errors. Errors can be discovered in a ConDec model using the mandatory automaton of that model. First, if there is no transition in the automaton that can be triggered by an activity then this activity is dead (cf. Definition 13). Second, if the automaton is empty (has no states and no transitions) then the model has a conflict (cf. Definition 14). It is possible to detect the smallest subset of constraints that causes the error by searching through the powerset of all mandatory constraints in the model. To

[2] Termination of instances is possible only if the automaton is in an accepting state (`s0` in our case).

[3] Note that Figure 5(b) shows a simplified deterministic automata for the response formula. The automata generated from LTL formulas are in general non-deterministic automata [13]. The standard determinization procedure [21] can be used to build a deterministic automaton.


[] (service -> (<> report))
response
service
report
not-response
[] (service -> (! <> report))
<> service
1..*

(a) “service” is a dead activity

(b) conflict

Fig. 6. Errors independent from history: dead activities and conflicts

achieve this, conjunction automata for subsets of constraints are created and analyzed. If an error is found in a subset, its supersets will be discarded during the search because all of them will contain the same error. This kind of verification can be performed on new models and during run-time changes (Section 4).

Activity “service” in the model given in Figure 6(a) is a dead activity due to the combination of the response and the not-response constraints, while other constraints in the model in Figure 4 do not contribute to this error. The response constraint expresses that every time “service” is executed, “register” has to be executed afterwards at least once. The not-response constraint specifies exactly the opposite, namely that activity “register” cannot be executed after activity “service”. As long as activity “service” is not executed in the model, both constraints are fulfilled. However, as soon as activity “service” is executed for the first time, it becomes impossible to fulfill both constraints. Therefore, we can not allow execution of activity “service” in any instance of this model.

Figure 6(b) shows a model with a conflicting combination of constraints. There is no possibility to satisfy: (1) the existence constraint (“1..*”) on “service”, (2) the response constraint on “service” and “report” and (3) the not-response constraint on “service” and “report”. Therefore, the combination of these three constraints causes a conflict.

3.2 Declare Tool

We developed the Declare tool for development and enactment of declarative process models. Declare can support various languages based on constraint templates as described in Section 3. Specifying languages in the tool is relatively easy – languages and constraint templates can be added, deleted, changed. Each language should include templates specific for a certain domain. For example, the DecSerFlow language [4] has been developed for web-service domain. This language is very similar to ConDec and it contains more than twenty constraint templates. Currently, Declare stores the semantics of constraint templates as LTL formulas, but it is implemented in a way that other formalization languages can be used. Templates are used in Declare to quickly define constraints in models: a template is first selected and activities form the model are assigned to the parameters of the template (cf. Figure 3). In this way, knowledge of the semantics formalization language (e.g., LTL) is not necessary for the development of models in Declare. Declare consists of three tools (see Figure 3.2): (1) the *Designer* is

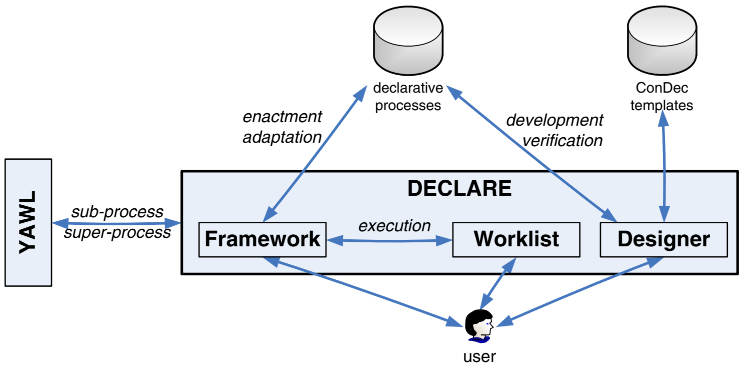


Fig. 7. Declare tool

used for the specification of languages (e.g. DecSerFlow, ConDec, etc.), specification of constraint templates, development of process models; (2) the *Framework* is the execution engine where process instances can be launched, run-time changes can be applied to instances, etc. and (3) the *Worklist* is a simple tool that each user uses to execute process instances. Declare works together with the YAWL workflow management system (`www.yawl-system.com`) [2]. On one hand, a DECLARE process can be implemented as a sub-process of a YAWL process. On the other hand, a YAWL process can be implemented as a sub-process of a DECLARE process. Therefore, it is possible to have workflows which are partly procedural and partly declarative. In the next section we show how Declare is extended to support changes.

4 Change in ConDec

Thanks to the usage of automata (cf. Section 3.1) it is fairly easy to change ConDec models for already running instances. ConDec supports both ad-hoc and evolutionary changes as defined in Section 2.3 (cf. Definitions 15, 16 and 17).

Procedure for change. The procedure for changing an instance during the execution is slightly different from the procedure for starting an instance. Figure 8 shows how the DECLARE tool performs both procedures. When an instance of a verified constraint model is started, an automaton is created for the mandatory constraints and is set to the initial state. After these steps, the instance starts executing with the constraint automaton in its initial state. During change, one additional step, called *instance verification* (cf. Definition 8), is needed to determine whether the history trace satisfies the constraints of the new (changed) model. Instance verification could reveal *history violations*. A history violation is a permanent violation of mandatory constraints of the new model. It occurs when the history (generated by the automaton for the old model) cannot be replayed by the new automaton (for the changed model). Change is only allowed in the absence of history violations. After change, the instance continues its execution according to the new model. The state of the new automaton is the state that was set by

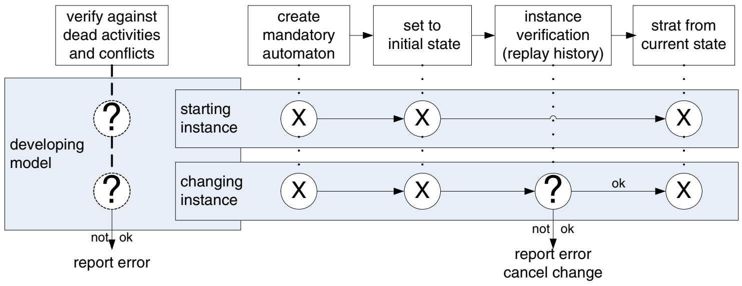


Fig. 8. Procedure for starting and changing instances

the history trace, i.e. history is “replayed” in the new model. In cases of history violation, the minimal subset of constraints causing the violation is detected using the same technique as we used for dead activities and conflict verification.

Change operations. Changes of ConDec models in the DECLARE tool can be achieved by: (1) adding constraints, (2) removing constraints, (3) adding activities and (4) removing activities. Using combinations of these four atomic change operations it is also possible to change constraint types from mandatory to optional and vice versa in running instances. For example, changing type of a constraint from optional into mandatory can be decomposed into two atomic actions: an optional constraint is removed and an mandatory constraint is added to the model.

The automaton used for execution of an instance is generated based on all mandatory constraints in the instance model. Therefore, this automaton will change only when adding/removing mandatory constraints. In cases of other changes (i.e., adding and removing activities and optional constraints), the execution automaton will remain the same like before the change. Due to this fact, history violations can only occur when adding mandatory constraints to the constraint model. When an activity is added (or removed) in the running instance, its execution automaton will remain the same, but the users will (or will not) be able to execute the activity in the future for the running instance (cf. Definition 9). When removing an activity involved in one or more constraints, one of the two strategies can be adopted: (1) the change operation is rejected and the activity cannot be removed or (2) the activity and all related constraints are removed from the model. Currently, the second strategy is implemented in Declare.

Figure 9(a) shows a ConDec model where a precedence constraint has been added to the model. The precedence constraint specifies that each “bless” activity can be executed only after at least one execution of activity “pray”. Figure 9(b) shows the automaton for all constraints of the ConDec model. In the automaton the “bless” activity is only allowed after execution of the “pray” activity (state `s1`). Figure 9(c) shows a history violation for trace ⟨curse, bless, pray, bless⟩. The violation is caused by the fact that the automaton is unable to execute “bless” in `s0`. Therefore, adding the “precedence” constraint is not allowed for instances with this history trace. Figure 10 shows a screenshot of Declare reporting this history violation for the “precedence” constraint.

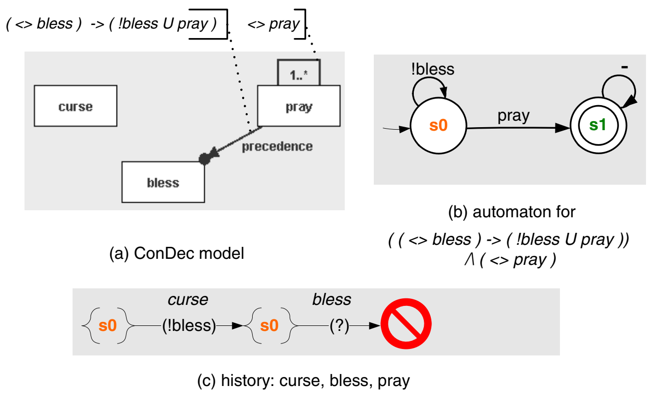
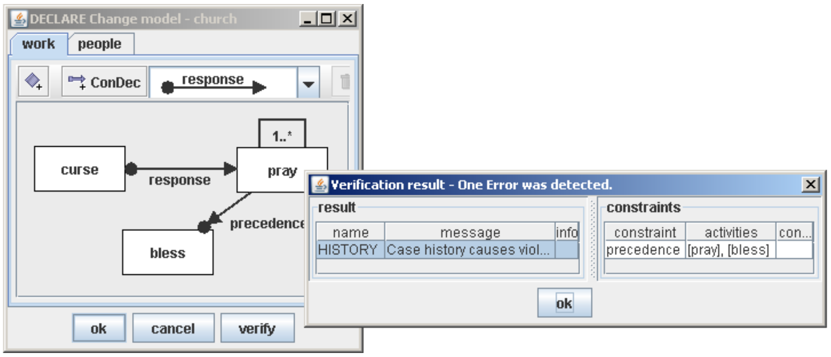


Fig. 9. "Precedence(pray,bless)" violates history ⟨curse, bless, pray⟩



name	message	info
HISTORY	Case history causes viol...	

constraint	activities	con...
precedence	[pray], [bless]	

Fig. 10. Declare screenshot - precedence constraint violating history ⟨curse, bless, pray⟩

Note that there might be cases where a group of constraint causes a history violation. Declare searches for the smallest subset of constraints that causes an verification error (dead activity, conflict or history violation) by searching through the powerset of all mandatory constraints as described in Section 3.1.

Change types. The DECLARE tool supports both total and partial evolutionary change of ConDec models. For an ad-hoc change, instance verification is only performed for the relevant instance. In traditional languages migration of instances is complicated and not always possible. The complexity of this operation stems from the fact that for the current state of an instance, an appropriate state in the new model has to be found and this is not always possible (cf. the "dynamic change bug" described in [11] and the many problems described in [19]). In declarative languages, such as ConDec, it is not necessary to find such a state. DECLARE only investigates the state of the new mandatory constraint automaton for the history trace(s), to detect whether migration is possible. For evolutionary changes, DECLARE performs instance verification on all instances of the old constraint model. Instances that do not cause history violation are migrated to the new model. Other instances continue execution with the old model.

5 Related Work

Many researchers have been trying to provide ways of avoiding the apparent paradox where, on the one hand, there is the desire to control the process and to avoid incorrect or undesirable executions of the processes, and, on the other hand, workers want lots of flexibly and to feel unconstrained in their actions [1,3,4,6,7,8,9,11,16,18,19,23]. It is impossible to provide a complete overview of related work. Therefore, we refer to only some of the most related papers in this area.

See [3] for a taxonomy of change and [12] for an introduction to the different types of workflow processes. The case handling concept is advocated as a way to avoid restricting users in their actions [6]. This is achieved by a range of mechanisms that allow for implicit deviations that are rather harmless. In [8] completely different techniques are used, but also the core idea is that implicit paths are generated to allow for more flexibility. In [7] pockets of flexibility are identified that are specified/selected later in the process, i.e., there is some form of "late binding" at run-time. Many papers look at problems related to ad-hoc and/or evolutionary change [1,9,11,18,19,23]. The problem of the dynamic change bug was introduced in [11]. In [1] this problem is addressed by calculating so-called change regions based on the structure of the process. A particular correctness property is described in [23] and the problem of instance migration is also investigated in [9]. In the context of the ADEPT system the problem of workflow change has been investigated in detail (including data analysis) [18,19].

It is also interesting to mention some commercial workflow management systems in this context. Historically, InConcert of Xerox and Ensemble of FileNet were systems among the first commercial systems to address the problem of change. Both supported ad-hoc changes in a rather restrictive setting. Several systems have been extended with some form of late binding. For example, the Staffware workflow system allows for the dynamic selection of subprocesses at run-time. Probably the most flexible commercial system is FLOWer of Pallas Athena [6]; this system supports a variety of case handling mechanisms to enable flexibility at run-time while avoiding changes of the model.

This paper is based on the earlier work on ConDec [16] and DecSerFlow [4] where a more declarative style of modeling is advocated. In those papers, the problem of change is not addressed, i.e., the goal is to avoid change. Despite various approaches to declarative (and constraint-based) workflow specification [10,22], as far as we know, this paper is the first paper that investigates the possibility of allowing ad-hoc and evolutionary changes in a constraint-based language.

6 Conclusions

This paper presented a new and comprehensive approach towards supporting change in constraint-based workflow models. This approach combines the advantages of having a declarative style of modeling and allowing ad-hoc and evolutionary changes. On the one hand, we try to avoid over-specification by using a declarative style of modeling rather than the typical procedural styles used in today's workflow management systems. On the other hand, we acknowledge the fact that sometimes change is unavoidable and provide extensive support for this. The results presented in this paper show that it is relatively easy to support ad-hoc and evolutionary changes in constraint-based workflow models.

In this paper, we presented a general approach and also showed a concrete application of the ideas using the ConDec language [16]. Moreover, the whole approach is supported by the Declare system. The reader is invited to download the tool from `http://is.tm.tue.nl/staff/mpesic/declare.htm`. Declare works together with the YAWL workflow management system [2] (`www.yawl-system.com`) that also allows for flexibility through so-called worklets [7]. This enables developing workflows which are partly procedural and partly declarative while using all kinds of flexibility mechanisms. Future work will aim at experimenting with interesting mixtures of these mechanisms, e.g., to provide guidelines on when to use particular types of flexibility.

References

1. van der Aalst, W.M.P.: Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change. Information Systems Frontiers 3(3), 297–317 (2001)
2. van der Aalst, W.M.P., Aldred, L., Dumas, M., ter Hofstede, A.H.M.: Design and Implementation of the YAWL System. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 142–159. Springer, Heidelberg (2004)
3. van der Aalst, W.M.P., Jablonski, S.: Dealing with Workflow Change: Identification of Issues and Solutions. International Journal of Computer Systems, Science, and Engineering 15(5), 267–276 (2000)
4. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer, Heidelberg (2006)
5. van der Aalst, W.M.P., Pesic, M.: Specifying, discovering, and monitoring service flows: Making web services process-aware. BPM Center Report BPM-06-09, BPM Center (2006), `http://www.BPMcenter.org`
6. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. Data and Knowledge Engineering 53(2), 129–162 (2005)
7. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. LNCS, vol. 4275, pp. 291–308. Springer, Heidelberg (2006)
8. Agostini, A., De Michelis, G.: Improving Flexibility of Workflow Management Systems. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 218–234. Springer, Heidelberg (2000)
9. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow Evolution. Data and Knowledge Engineering 24(3), 211–238 (1998)
10. Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P., Zbyslaw, A.: Freeflow: mediating between representation and action in workflow systems. In: CSCW 1996. Proceedings of the 1996 ACM conference on Computer supported cooperative work, pp. 190–198. ACM Press, New York, NY, USA (1996)

11. Ellis, C.A., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: Comstock, N., Ellis, C., Kling, R., Mylopoulos, J., Kaplan, S. (eds.) ACM SIGOIS. Proceedings of the Conference on Organizational Computing Systems, Milpitas, California, pp. 10–21. ACM Press, New York (1995)
12. Georgakopoulos, D., Hornick, M., Sheth, A.: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. Distributed and Parallel Databases 3, 119–153 (1995)
13. Giannakopoulou, D., Havelund, K.: Automata-based verification of temporal properties on running programs. In: ASE 2001. Proceedings of the 16th IEEE international conference on Automated software engineering, p. 412. IEEE Computer Society Press, Washington, DC, USA (2001)
14. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge, Massachusetts and London, UK (1999)
15. Milner, R.: Communicating and Mobile Systems: The Pi-Calculus. Cambridge University Press, Cambridge, UK (1999)
16. Pesic, M., van der Aalst, W.M.P.: A Declarative Approach for Flexible Business Processes. In: Eder, J., Dustdar, S. (eds.) Business Process Management Workshops. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006)
17. Petri, C.A.: Kommunikation mit Automaten. PhD thesis, Fakultät für Mathematik und Physik, Technische Hochschule Darmstadt, Darmstadt, Germany (1962)
18. Reichert, M., Dadam, P.: ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. Journal of Intelligent Information Systems 10(2), 93–129 (1998)
19. Rinderle, S., Reichert, M., Dadam, P.: Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. Data and Knowledge Engineering 50(1), 9–34 (2004)
20. Scheer, A.-W.: ARIS: business process modeling, 2nd edn. Springer, Berlin (1998)
21. Sudkamp, T.-A.: Languages and machines: an introduction to the theory of computer science, 2nd edn. Addison-Wesley Pub., Reading (1997)
22. Wainer, J., de Lima Bezerra, F.: Groupware: Design, Implementation, and Use. In: Favela, J., Decouchant, D. (eds.) CRIWG 2003. LNCS, vol. 2806, pp. 151–158. Springer, Heidelberg (2003)
23. Weske, M.: Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In: Sprague, R. (ed.) HICSS-34. Proceedings of the Thirty-Fourth Annual Hawaii International Conference on System Science, IEEE Computer Society Press, Los Alamitos, California (2001)

Dynamic, Extensible and Context-Aware Exception Handling for Workflows

Michael Adams[1], Arthur H.M. ter Hofstede[1], and Wil M.P. van der Aalst[1,2], and David Edmond[1]

[1] Business Process Management Group
Queensland University of Technology, Brisbane, Australia
{m3.adams, a.terhofstede, d.edmond}@qut.edu.au
[2] Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
w.m.p.v.d.aalst@tue.nl

Abstract. This paper presents the realisation, using a Service Oriented Architecture, of an approach for dynamic, flexible and extensible exception handling in workflows, based not on proprietary frameworks, but on accepted ideas of how people actually work. The resultant service implements a detailed taxonomy of workflow exception patterns to provide an extensible repertoire of self-contained exception-handling processes called *exlets*, which may be applied at the task, case or specification levels. When an exception occurs at runtime, an exlet is dynamically selected from the repertoire depending on the context of the exception and of the particular work instance. Both expected and unexpected exceptions are catered for in real time, so that 'manual handling' is avoided.

1 Introduction

Workflow management systems (WfMS) are used to configure and control structured business processes from which well-defined workflow models and instances can be derived [1,2,3]. However, the proprietary process definition frameworks imposed by WfMSs make it difficult to support (i) dynamic evolution (i.e. modifying process definitions during execution) following unexpected or developmental change in the business processes being modelled [4]; and (ii) process exceptions, or deviations from the prescribed process model at runtime [5,6].

For exceptions, the accepted practice is that if it can conceivably be anticipated, then it should be included in the static process model. However, this approach can lead to very complex models, much of which will not be executed in most instances. Also, mixing business logic with exception handling routines complicates the verification and modification of both [7], in addition to rendering the process model almost unintelligible to many stakeholders.

Conversely, if an *unexpected* exception occurs, then the model is deemed to be simply deficient and thus must be amended to include the previously unimagined event (see for example [8]), which disregards the frequency of such events and the costs involved with their correction. Most often, the only available options are suspension while the exception is handled manually or termination of the case, but since most processes are long and complex, neither option presents a satisfactory solution [7]. Manual handling incurs an added penalty: the corrective actions undertaken are not added to 'organisational memory' [9,10], and so natural process evolution is not incorporated into future iterations of the process. Associated problems include those of migration, synchronisation and version control [5].

Thus a large group of business processes do not easily map to the rigid modelling structures provided [11], due to the lack of flexibility inherent in a framework that, by definition, imposes rigidity. Business processes are 'system-centric', or *straight-jacketed* [2] into the supplied framework, rather than truly reflecting the way work is actually performed [1]. As a result, users are forced to work outside of the system, and/or constantly revise the static process model, in order to successfully perform their activities, thereby negating the efficiency gains sought by implementing a workflow solution in the first place.

The flux inherent in work practices has been further borne out by our previous work on process mining. When considering processes where people are expected to execute tasks in a structured way but are not forced to by a workflow system, process mining shows that the processes are much more dynamic than expected. That is, workers tend to deviate from the 'normal flow', often with good reasons.

To gain a grounded understanding of actual work practices, we previously undertook a detailed study of *Activity Theory*, a broad collective of theorising and research in organised human activity (cf. [12,13]), and derived from it a set of principles that describe the nature of participation in organisational work practices [14]. We then applied those principles to the design and implementation of a discrete web-based service that maintains an extensible repertoire of self-contained exception handling processes, known as *exlets*, and an associated set of contextual selection rules, to dynamically support exception handling for business process instances orthogonal to the underlying workflow engine.

This paper describes the resultant service, which uses the *'worklets'* approach introduced in [15,16] as a conceptual foundation, and applies the classification of workflow exception patterns from [17]. The implementation platform used is the well-known workflow environment YAWL [18,19], which supports a Service Oriented Architecture (SOA) — but the discrete nature of the service means its applicability is in no way limited to the YAWL environment. Also, being open-source, the service is freely available for use and extension.

The paper is organised as follows: Section 2 provides an overview of the design and operation of the service, while Section 3 details the service architecture. Section 4 discusses exception types handled by the service and the definition of exlets. Section 5 describes how the approach utilises *Ripple Down Rules* (RDR) to achieve contextual, dynamic selection of exlets at runtime. Section 6 discusses related work, and finally Section 7 concludes the paper.

2 Service Overview

The implemented service, known as the *Worklet Service*[1] comprises two distinct but complementary sub-services: a *Selection sub-service*, which enables dynamic flexibility for process instances (cf. [15]); and an *Exception Handling sub-service* (the subject of this paper), which provides facilities to handle both expected and unexpected process exceptions at runtime.

The Exception Handling sub-service (or, more simply, the *Exception Service*) allows administrators to define exception handling processes (called *exlets*) for parent workflow instances, to be invoked when certain events occur, and thereby allowing execution of the parent process to continue unhindered. It has been designed so that the enactment engine, besides providing notifications at certain points in the life cycle of a process instance, needs no knowledge of an exception occurring, nor of any consequent invocation of exlets — all exception checking and handling is provided by the service. Additionally, all exlets in a specification's repertoire automatically become an implicit part of the process specification for all current and future instances of the process, which provides for continuous evolution of the process while avoiding any requirement to modify the original process definition.

The Exception Service is built on the same conceptual framework as the Worklet Selection sub-service, and so uses the same repertoire and dynamic rules approach (see Section 5). There are, however, two fundamental differences between the two sub-services. First, where the Selection Service selects a worklet as the result of satisfying a rule in a rule set, the result of an Exception Service rule being satisfied is an exlet (which may contain a worklet to be executed as a compensation process). Second, while the Selection Service is invoked for certain nominated tasks in a process, the Exception Service, when enabled, is invoked for *every* case and task executed by the enactment engine, and will detect and handle up to ten different kinds of process exceptions (those exception types are described in Section 4.1).

Most modern programming languages provide mechanisms that separate exception handling routines from the 'normal' program logic, which facilitates the design of readable, comprehensible programs [20,21,22]. Similar methods are incorporated into distributed frameworks and operating systems. However, little or no such means are provided in most WfMSs. Usually, any or all possible exceptions must be incorporated into the monolithic workflow model, which contravenes accepted paradigms of modularity, encapsulation and reusability.

For the Exception Service, an exlet (discrete and external to the parent model) may consist of a number of various actions (such as cancel, suspend, complete, fail, restart and compensate) and be automatically applied at a workitem, case and/or specification level. And, because exlets can include worklets as compensation processes, the original parent process model only needs to reveal the actual business logic for the process.

[1] Essentially, a *worklet* is a small, discrete workflow process that may act as both a late-bound sub-net for an enabled workitem and a compensation process within an exception handler.

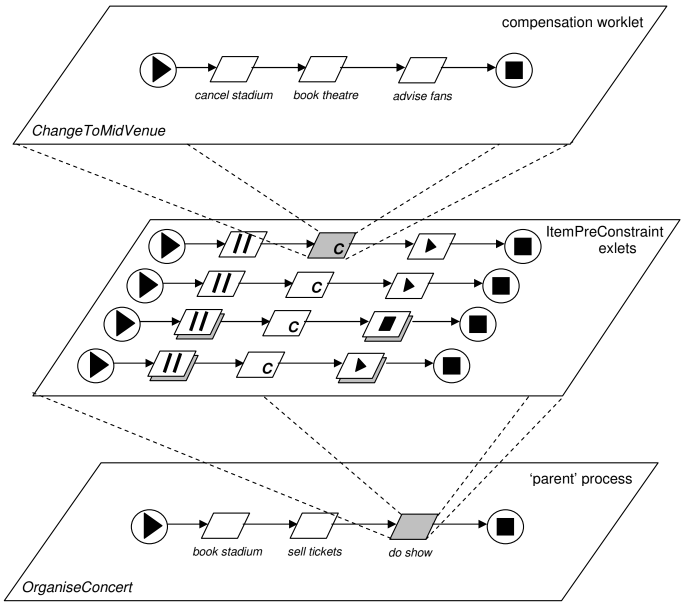


Fig. 1. Process – Exlet – Worklet Hierarchy

Each time an exception occurs, the service makes a choice from the repertoire based on the type of exception and the contextual data of the workitem/case, using a set of rules to select the most appropriate exlet to execute (see Section 5). If the exlet contains a compensation primitive, the associated worklet is run as a separate case in the enactment engine, so that from an engine perspective, the worklet and its 'parent' (i.e. the process that invoked the exception) are two distinct, unrelated cases. The service tracks the relationships, data mappings and synchronisations between cases, and maintains execution logs that may be combined with those of the engine via case identifiers to provide a complete operational history of each process. Figure 1 shows the relationship between a 'parent' process, an exlet repertoire and a compensatory worklet, using as an example a simple process for the organisation of a rock concert (*Organise Concert*).

The repertoire of exlets grows as new exceptions arise or different ways of handling exceptions are formulated, *including while the parent process is executing*, and those handling methods automatically become an implicit part of the process specification for all current and future instances of the process.

Any number of exlets can form the repertoire of each particular exception type for an individual task or case. An exlet may be a member of one or more repertoires – that is, it may be re-used for several distinct tasks or cases within and across process specifications. The Selection and Exception sub-services can be used in combination within case instances to achieve dynamic flexibility *and* exception handling simultaneously.

3 Service Architecture

The Worklet Service has been implemented as a YAWL Custom Service [18,19]. The YAWL environment was chosen as the implementation platform since it provides a very powerful and expressive workflow language based on the workflow patterns identified in [23], together with a formal semantics. It also provides a workflow enactment engine, and an editor for process model creation, that support the control flow, data and (basic) resource perspectives.

The YAWL environment is open-source and offers a service-oriented architecture, allowing the service to be implemented completely independent to the core engine. Thus the deployment of the Worklet Service *is in no way limited* to the YAWL environment, but may be ported to other environments (for example, BPEL based systems, classical workflow systems, and the Windows Workflow Foundation) by making the necessary linkages in the service interface. As such, this implementation also represents a case study in service-oriented computing whereby dynamic flexibility and exception handling for workflows, orthogonal to the underlying workflow language, is provided.

To enable the Worklet Service to serve a workflow enactment engine, a number of events and methods must be provided by an interface between them. In the conceptualisation and design of the service, the size or 'footprint' of the interface has been kept to an absolute minimum to accommodate ease-of-deployment and thus maximise the installation base, or the number of enactment engines, that may benefit from the extended capabilities that the worklet service offers. Being a web-service, the worklet service has been designed to enable remote deployment and to allow a single instance of the service to concurrently manage the flexibility and exception handling management needs for a number of disparate enactment engines that conform to the interface.

The YAWL environment provides for the workflow enactment engine and external services to interact across several interfaces supporting the ability to send and receive both messages and XML data to and from the engine. Three of those interfaces are used by the Worklet Exception Service:

– *Interface A* provides endpoints for process definition, administration and monitoring [19] – the service uses Interface A to upload worklet specifications to the engine;
– *Interface B* provides endpoints for client and invoked applications and workflow interoperability [19] – used by the service for connecting to the engine, to start and cancel case instances, and to check workitems in and out of the engine after interrogating their associated data; and
– *Interface X* ('X' for 'eXception') which has been designed to provide the engine with the ability to notify custom services of certain events and checkpoints during the life-cycle of each process instance and each of its tasks where process exceptions either may have occurred or should be tested for. Thus Interface X provides the Exception Service with the necessary triggers to dynamically capture and handle process exceptions.

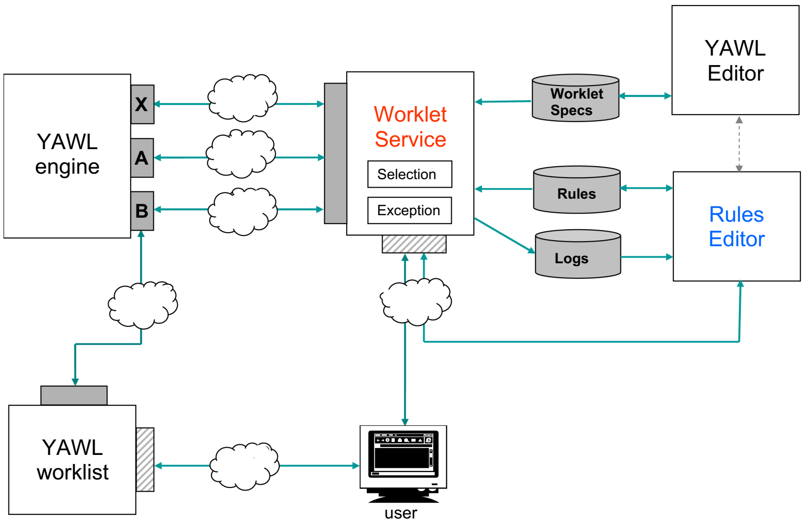


Fig. 2. External Architecture of the Worklet Service

In fact, Interface X was created to enable the Exception Service to be built. However, one of the overriding design objectives was that the interface should be structured for generic application — that is, it can be applied by a variety of services that wish to make use of checkpoint and/or event notifications during process executions. For example, in addition to exception handling, the interface's methods provide the tools to enable ad-hoc or permanent adaptations to process schemas, such as re-doing, skipping, replacing and looping of tasks.

Figure 2 shows the external architecture of the Worklet Service. The entities 'Worklet specs', 'Rules' and 'Logs' in Figure 2 comprise the *worklet repository.* The service uses the repository to store rule sets, worklet specifications for uploading to the engine, and generated process and audit logs. The YAWL editor is used to create new worklet specifications, and may be invoked from the Rules Editor, which is used to create new or augment existing rule sets, making use of certain selection logs to do so, and may communicate with the Worklet Service via a JSP/Servlet interface to override worklet selections following rule set additions (see Section 5). The service also provides servlet pages that allow users to directly communicate with the service to raise external exceptions and carry out administration tasks.

4 Exception Types and Handling Primitives

This section introduces the ten different types of process exception that have been identified, seven of which are supported by the current version of the Exception Service. It then describes the handling primitives that may be used to form an exception handling process (i.e. an exlet). The exception types and primitives described here are based on and extend from those identified by Russell et al., who define a rigorous classification framework for workflow exception handling independent of specific modelling approaches or technologies [17].

4.1 Exception Types

The following seven types of exceptions are supported by our current implementation:

Constraint Types: Constraints are rules that are applied to a workitem or case immediately before and after execution of that workitem or case. Thus, there are four types of constraint exception:

– *CasePreConstraint* - case-level pre-constraint rules are checked when each case instance begins execution;
– *ItemPreConstraint* - item-level pre-constraint rules are checked when each workitem in a case becomes enabled (i.e. ready to be checked out);
– *ItemPostConstraint* - item-level post-constraint rules are checked when each workitem moves to a completed status; and
– *CasePostConstraint* - case-level post constraint rules are checked when a case completes.

When the service receives an constraint event notification, the rule set is queried (see Section 5), and if a constraint has been violated the associated exlet is selected and invoked.

TimeOut: A timeout event occurs when a workitem reaches a set deadline. The service receives a reference to the workitem and to each of the other workitems running in parallel to it. Therefore, timeout rules may be defined for each of the workitems affected by the timeout (including the actual timed out workitem itself).

Externally Triggered Types: Externally triggered exceptions occur because of an occurrence outside of the process instance that has an effect on the continuing execution of the process. Thus, these events are triggered directly by a user via a servlet page (for example, Figure 5); depending on the actual event and the context of the case or workitem, a particular exlet will be invoked. There are two types of external exceptions, CaseExternalTrigger (for case-level events) and ItemExternalTrigger (for item-level events).

Three more exception types have been identified but are not yet supported, since they rely more heavily on the internal mechanisms of the enactment engine:

ItemAbort: This event occurs when a workitem being handled by an external program (as opposed to a human user) reports that the program has aborted before completion.

ResourceUnavailable: This event occurs when an attempt has been made to allocate a workitem to a resource and the resource reports that it is unable to accept the allocation or the allocation cannot proceed.

ConstraintViolation: This event occurs when a data constraint has been violated for a workitem *during* its execution (as opposed to pre- or post- execution).

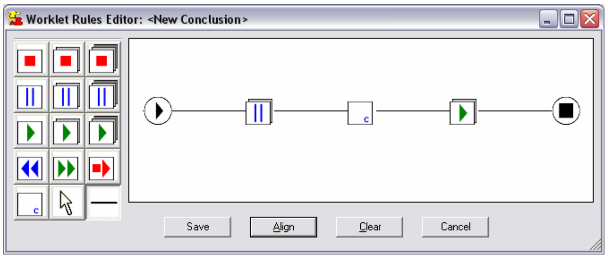


Fig. 3. Example Exlet in the Rules Editor

4.2 Exception Handling Primitives

Each exlet is defined graphically using the Worklet Rules Editor, and may contain any number of steps, or *primitives*. Figure 3 shows the Rules Editor with an example exlet displayed. On the left of the Editor is the set of available primitives, which are (reading left-to-right, top-to-bottom):

- *Remove WorkItem*: removes (or cancels) the workitem; execution ends, and the workitem is marked with a status of cancelled. No further execution occurs on the process path that contains the workitem.
- *Remove Case*: removes the case. Case execution ends.
- *Remove All Cases*: removes all case instances for the specification in which the workitem is defined, or of which the case is an instance.
- *Suspend WorkItem*: suspends (or pauses) execution of a workitem, until it is continued, restarted, cancelled, failed or completed, or the case that contains the workitem is cancelled or completed.
- *Suspend Case*: suspends all 'live' workitems in the current case instance (a live workitem has a status of fired, enabled or executing), effectively suspending execution of the entire case.
- *Suspend All Cases*: suspends all 'live' workitems in all of the currently executing instances of the specification in which the workitem is defined, effectively suspending all running cases of the specification.
- *Continue WorkItem*: un-suspends (or continues) execution of the previously suspended workitem.
- *Continue Case*: un-suspends execution of all previously suspended workitems for the case, effectively continuing case execution.
- *Continue All Cases*: un-suspends execution of all workitems previously suspended for all cases of the specification in which the workitem is defined or of which the case is an instance, effectively continuing all previously suspended cases of the specification.
- *Restart WorkItem*: rewinds workitem execution back to its start. Resets the workitem's data values to those it had when it began execution.
- *Force Complete WorkItem*: completes a 'live' workitem. Execution of the workitem ends, and the workitem is marked with a status of *ForcedComplete*,

which is regarded as a successful completion, rather than a cancellation or failure. Execution proceeds to the next workitem on the process path.
– *Force Fail WorkItem*: fails a 'live' workitem. Execution of the workitem ends, and the workitem is marked with a status of *Failed*, which is regarded as an unsuccessful completion, but not as a cancellation – execution proceeds to the next workitem on the process path.
– *Compensate*: runs a compensatory process (i.e. a worklet). Depending on the actions of previous primitives in the exlet, the worklet may execute simultaneously to the parent case, or execute while the parent is suspended. One or more worklets may be simultaneously invoked by a compensate primitive.

Thus, the example exlet in Figure 3 will suspend the case, execute a compensation process, then continue (or unsuspend) the case. A compensation primitive may contain an array of one or more worklets – when multiple worklets are defined they are launched concurrently as an effectively composite compensatory action. Execution moves to the next primitive in the exlet when all worklets have completed. Additionally, relevant data values may be mapped from a case to a compensatory worklet, where they may be modified and mapped back again to the original case.

Worklets that are executed as compensatory processes within exlets can in turn invoke child worklets to any depth. The primitives 'Suspend All Cases', 'Continue All Cases' and 'Remove All Cases' may be flagged when being added to an exlet definition in the Rules Editor so that their action is restricted to ancestor cases only. Ancestor cases are those in a hierarchy of worklets back to the original parent case — that is, where a process invokes an exlet which invokes a compensatory worklet which in turn invokes another worklet and/or an exlet, and so on. Since compensatory worklets are launched as separate cases in the enactment engine, they too are monitored by the service for exceptions and thus may have exlets launched for them in certain circumstances. Also, the 'Continue' primitives are applied only to those workitems and cases that were previously suspended by the same exlet. Execution moves to the next primitive in the exlet when all worklets launched from a compensation primitive have completed.

Referring to Figure 1, the centre tier shows the exlets repertoire for an ItemPreConstraint violation for a particular task, which correspond to the rule tree shown in Figure 4. There may actually be up to eleven different 'planes' for this tier — one for each exception type. Also, each exlet may refer to a different set of compensatory processes, or worklets, and so at any point there may be several worklets operating on the upper tier.

5 Contextual Selection of Exlets

The runtime selection of an exlet relies on the type of exception that has occurred and the relevant context of the workitem and/or case instance, derived from data attribute values of the case instance, workitem-level values, the internal status of each workitem in the process instance, resource data, historical data from process logs, and other extensible external sources. Some of these data are supplied directly by the enactment engine across the interfaces, others may be indirectly supplied using process mining techniques.

The selection process is achieved through the use of modified *Ripple Down Rules* (RDR), which comprise a hierarchical set of rules with associated exceptions, first devised by Compton and Jansen [24]. The fundamental feature of RDR is that it avoids the difficulties inherent in attempting to compile, *a-priori*, a systematic understanding, organisation and assembly of all knowledge in a particular domain. Instead, it allows for general rules to be defined first with refinements added later as the need arises [25].

Each specification may have an associated rule set, which consists of a set of RDR trees stored as XML data. Each RDR tree is a collection of simple rules of the form "if *condition* then *conclusion*", conceptually arranged in a binary tree structure (see Figure 4). When a rule tree is queried, it is traversed from the root node of the tree along the branches, each node having its condition evaluated along the way. For non-terminal nodes, if a node's condition evaluates to *True*, the node connected on its *True* branch is subsequently evaluated; if it evaluates to *False*, the node connected on its *False* branch is evaluated [26]. When a terminal node is reached, if its condition evaluates to *True* then that conclusion is returned as the result of the tree traversal; if it evaluates to *False*, then the conclusion of the last node in the traversal that evaluated to *True* is returned as the result.

Effectively, each rule node on the true branch of its parent is an exception rule of the more general one of its parent (that is, it is a *refinement* of the more general parent rule), while each rule node on the false branch of its parent node is an "else" rule to its parent (or an *alternate* to the parent rule). This tree traversal provides implied *locality* - a rule on an exception branch is tested for applicability only if its parent (next-general) rule is also applicable.

The hierarchy of a worklet rule set is (from the bottom up):

– **Rule Node:** contains the details (condition, conclusion, id, parent and so on) of one discrete ripple-down rule. The conclusion of a node equates to an exlet.
– **Rule Tree:** consists of a number of rule nodes conceptually linked in a binary tree structure.
– **Tree Set:** a set of one or more rule trees. Each tree set is specific to a particular exception type. The tree set of a case-level exception type will contain exactly one tree. The tree set of an item-level type will contain one rule tree for each task of the specification that has rules defined for it.
– **Rule Set:** a set of one or more tree sets representing the entire set of rules defined for a specification. Each rule set is specific to a particular specification. A rule set will contain one tree set for each exception type for which rules have been defined.

A repertoire of exlets may be formed for each exception type. Each specification has a unique rule set (if any), which contains between one and eleven tree

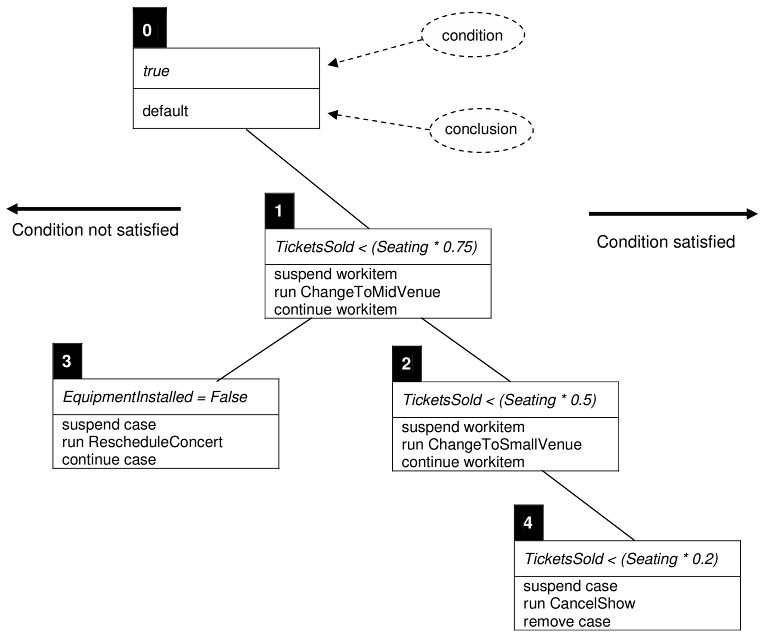
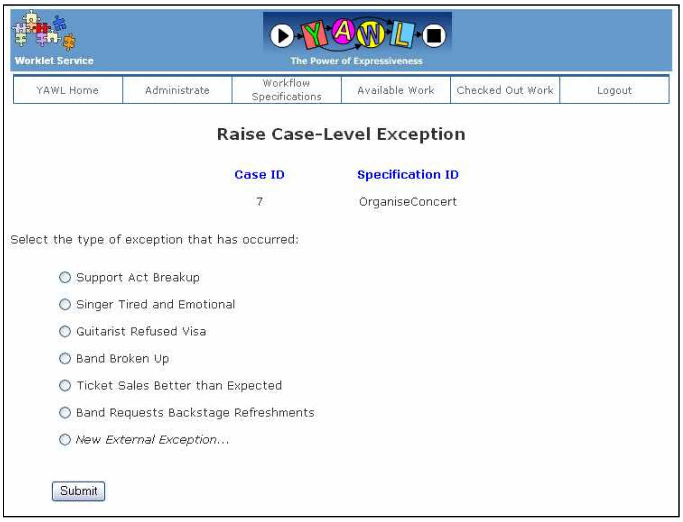


Fig. 4. Example rule tree (ItemPreConstraint for DoShow task of OrganiseConcert)

sets (or sets of rule trees), one for selection rules (used by the Selection sub-service) and one for each of the ten exception types. Three of those ten relate to case-level exceptions (i.e. CasePreConstraint, CasePostConstraint and CaseExternalTrigger) and so each of these will have at most one rule tree in the tree set. The other eight tree sets relate to workitem-level events (seven exception types plus selection), and so may have one rule tree for each task in the specification — that is, the tree sets for these eight rule types may consist of a number of rule trees. The rule set for each specification is stored as XML data in a discrete disk file. All rule set files are stored in the worklet repository.

If there are no rules defined for a certain exception type in the rule set for a specification, a runtime event of that type is ignored by the service. Thus rules are needed only for those exception events that are desired to be handled for a particular task and/or specification. So, for example, if an administrator is interested only in capturing pre- and post- constraints at the workitem level, then only the ItemPreConstraint and ItemPostConstraint tree sets need to be defined (that is, rules defined within those tree sets). Of course, rules for other types can be added later when required. Figure 4 shows the ItemPreConstraint rule tree for the third task in the Organise Concert example, *Do Show*, (corresponding to the centre and lower tiers of Figure 1 respectively); it is evaluated when a *Do Show* workitem instance is enabled. This rule tree provides exlets for organisers to change the venue of the concert, or cancel it, when there are insufficient tickets sold to fill the original venue. For example, if a particular *Do Show* instance has



Raise Case-Level Exception

Case ID	Specification ID
7	OrganiseConcert

Select the type of exception that has occurred:

- ○ Support Act Breakup
- ○ Singer Tired and Emotional
- ○ Guitarist Refused Visa
- ○ Band Broken Up
- ○ Ticket Sales Better than Expected
- ○ Band Requests Backstage Refreshments
- ○ *New External Exception...*

Submit

Fig. 5. Raise Case-Level Exception Screen (Organise Concert example)

a value for the attribute ‘TicketsSold’ that is less than 75% of the attribute ‘Seating’ (i.e. the seating capacity of the venue), an exlet is run that suspends the workitem, runs the compensatory worklet ChangeToMidVenue, and then, once the worklet has completed, continues (or unsuspends) the workitem. By following the exception path of the rule tree, it can be seen that each subsequent node is a refinement of its parent, since it is only evaluated if its parent rule is satisfied. So, if the tickets sold are also less than 50% of the capacity, then we want instead to suspend the workitem, run the ChangeToSmallVenue worklet, and then unsuspend the workitem. Finally, if less than 20% of the tickets have been sold, we want to suspend the entire case, run a worklet to perform the tasks required to cancel the show, and then remove (i.e. cancel) the case[2].

As mentioned previously, the service provides a set of servlet pages that can be invoked directly by the user via add-ins to the YAWL worklist handler, which are visible only when the service is enabled. One of the servlet pages allows a user to raise an exception directly with the service (i.e. bypassing the engine) at any time during the execution of a case. When invoked, the service lists from the rule set for the selected case the existing external exception triggers (if any) for the case’s specification (see Figure 5). Note that these triggers may describe events that may be considered either adverse (e.g. Band Broken Up) *or* beneficial (e.g. Ticket Sales Better Than Expected) to the current case, or may simply represent new or additional tasks that need to be carried out for the particular case instance (e.g. Band Requests Backstage Refreshments). When a trigger is selected by the

[2] It has been formally shown that an RDR tree traverses through a smaller number of rules enroute to its final conclusion than traversal through an equivalent decision list [25].

user, the corresponding rule set is queried and the appropriate exlet, relative to the case's context and the trigger selected, is executed. Item-level external exceptions can be raised in a similar way.

Notice that at the bottom of the list of triggers in Figure 5 the option to add a *New External Exception* is provided. If an unexpected external exception arises that none of the available triggers represent, a user can use that option to notify an administrator, via another servlet page, of the new exception, its context and possible ways to handle it — the notification of an unexpected external exception automatically suspends the case as a safeguard. The administrator can then create a new exlet in the Rules Editor and, from the Editor, connect directly to the service to launch the new exlet for the parent case. New exlets for unexpected internal exceptions are raised and launched using the same approach as that described for the Selection sub-service in [15].

The examples used in this section have been intentionally simplified to demonstrate the operation of the Exception Service; while not intended to portray a realistic process, it is desirable to not camouflage the subject of this paper by using a more realistic, and thus a necessarily more complex process. However, exemplary studies have been undertaken using real-world processes from both a relatively rigid business scenario and a more creative environment, which serve to fully validate the approach (see Chapter 7 of [27]).

6 Related Work

Since the mid-nineties much research has been carried out on issues related to exception handling in workflow management systems. Such research was initiated because, generally, commercial workflow management systems provide only basic support for handling exceptions [17,28,7,29] (besides modelling them directly in the main 'business logic'), and each deals with them in a proprietary manner; they typically require the model to be fully defined before it can be instantiated, and changes must be incorporated by modifying the model statically.

While it is not the intention of this paper to provide a complete overview of the work done in this area, reference is made here to a number of quite different approaches. For a more systematic overview see [17], where different tools are evaluated with respect to their exception handing capabilities using a patterns-based approach.

Tibco iProcess provides constructs called *event nodes*, from which a separate pre-defined exception handling path or sequence can be activated when an exception occurs. It may also suspend a process either indefinitely or wait until a timeout occurs. If a work item cannot be processed it is forwarded to a 'default exception queue' where it may be manually purged or re-submitted. *COSA* provides for the definition of external 'triggers' or events that may be used to start a sub-process. All events and sub-processes must be defined at design time. *Websphere MQ Workflow* supports timeouts and, when they occur, will branch to a pre-defined exception path and/or send a message to an administrator. SAP Workflow supports exception events for cancelling workflow instances, for checking workitem pre- and post- constraints, and for 'waiting' until an external trigger occurs. Exception handling processes may be assigned to a workflow based on the type of exception that has occurred, although the handlers for each are specified at design time, and only one may be assigned to each type. FLOWer is described as a 'case-handling' system, and supports some exception handling actions [2]. For example, a deadline expiry can automatically complete a workitem. Also, some support for constraint violation is offered: a plan may be automatically created or completed when a specified condition evaluates to true [17].

Among the non-commercial systems, the *OPERA* prototype [7] has a modular structure in which activities are nested. When a task fails, its execution is stopped and the control of the process is handed over to a single handler predefined for that type of exception — the context of the activity is not accounted for. If the handler cannot solve the problem, it propagates the exception up the activity tree; if no handler can be found the entire process instance aborts. The *eFlow* system [30] supports the definition of compensation rules for regions, although they are static and cannot be defined separately to the standard model. *AgentWork* [31] provides the ability to modify process instances by dropping and adding individual tasks based on events and ECA rules. However, the rules do not offer the flexibility or extensibility of Ripple Down Rules, and changes are limited to individual tasks, rather than the task-process-specification hierarchy supported by the Worklet Service. Also, the possibility exists for conflicting rules to generate incompatible actions, which requires manual intervention and resolution. The *TREX* system [32] allows the modeller to choose from a catalog of exception handlers during runtime to handle exceptions as they arise; however, it requires a human agent to intervene whenever an exception occurs. Also, the exceptions handled are, for the most part, transactional, and scope for most expected and all unexpected exceptions is not provided. The *MARIFlow* system [33] supports document exchange and coordination across the internet. The system supports some transactional-level exception handling, for example rolling back and restarting a blocked or dead process, but there is no support for dynamic change or handling exceptions within the control flow of the process. *CBRFlow* [34] uses a case-based reasoning approach to support adaptation of predefined workflow models to changing circumstances by allowing (manual) annotation of business rules during run-time via incremental evaluation. It should be noted that only a small number of academic prototypes have had any impact on the frameworks offered by commercial systems [17,28].

The majority of languages used to described and define business process models are of a procedural nature, which limits their effectiveness in very flexible environments [35]. For example, BPEL provides *compensation handlers* that are intended to support rollback or undo of activities after an error has occurred; however, they are essentially unable to access the current process state [36] — thus the context of the case cannot be taken into account. In addition, compensation handlers cannot affect other process instances (i.e. at the specification level), cannot be used to effect non-erroneous changes in process execution [36] and may only perform a termination action (all in contrast to the various actions supported by the Worklet Exception Service). Also, BPEL offers no support for constraint violations [17].

In summary, approaches to workflow flexibility and exception handling usually rely on a high-level of runtime user interactivity, which directly impedes on the basic aim of workflow systems (to bring greater efficiencies to work practices) and distracts users from their primary work procedures into process support activities. Another common theme is the complex update, modification and migration issues required to evolve process models. The Worklet Service differs considerably from those approaches. Exlets, that may include worklets as compensatory processes, as members of a repertoire, and dynamically linked to Ripple Down Rule sets, provide a novel, complete and extensible approach for the provision of dynamic exception handling in workflows.

7 Conclusion

The Worklet Exception Service has been constructed around the idea that exceptions, or deviations from a process specification, are a natural occurrence within almost every instantiation of a process. Thus the service provides a fully featured exception handling paradigm that detects (through constraint checking), reacts to, handles and incorporates exceptions and the way they are handled as they occur. The service also allows for unexpected exceptions to be handled during execution, so that a process instance need not be terminated when one occurs, or be handled off-system.

The service provides easy to use mechanisms to incorporate new handling procedures for unexpected exceptions implicitly into the process specification so that they are automatically available for all current and future instantiations of the specification. Thus a repertoire of exception handling procedures is maintained by the service for each process specification, so completely avoiding the need to modify a specification each time a deviation from its prescribed flow occurs — which also avoids the on-costs associated with taking the specification off-line while modifications are performed and verified, versioning problems, migration control issues and so on.

In providing these benefits, the Worklet Exception Service:

- Keeps the parent model clean and relatively simple;
- Promotes the reuse of sub-processes in different models, and allows standard processes to also be used as compensation processes, and vice versa;
- Maintains an extensible repertoire of exlets that can be constructed during design and/or runtime and can be invoked as required;
- Allows a specification to implicitly build a history of executions, providing for a learning system that can take the appropriate actions within certain contexts automatically;
- Maintains exlets, and compensatory worklets, as fully encapsulated processes, which allows for easier verification and modification; and

– Allows a model to evolve without the need to stop and modify the design of the whole specification when an exception occurs.

All system files, source code and documentation for YAWL and the worklet service, including the examples discussed in this paper, may be downloaded via `www.yawl-system.com`.

References

1. Bider, I.: Masking flexibility behind rigidity: Notes on how much flexibility people are willing to cope with. In: Castro, J., Teniente, E. (eds.) CAiSE 2005 Workshops, vol. 1, pp. 7–18, FEUP Edicoes, Porto (2005)
2. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: A new paradigm for business process support. Data & Knowledge Engineering 53(2), 129–162 (2005)
3. Joeris, G.: Defining flexible workflow execution behaviors. In: Dadam, P., Reichert, M. (eds.) Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. CEUR Workshop Proceedings, Paderborn, Germany, vol. 24, pp. 49–55 (October 1999)
4. Borgida, A., Murata, T.: Tolerating exceptions in workflows: a unified framework for data and processes. In: WACC 1999. Proceedings of the International Joint Conference on Work Activities, Coordination and Collaboration, pp. 59–68. ACM Press, San Francisco, California, USA (1999)
5. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems: a survey. Data and Knowledge Engineering 50(1), 9–34 (2004)
6. Casati, F.: A discussion on approaches to handling exceptions in workflows. In: Proceedings of the CSCW Workshop on Adaptive Workflow Systems, Seattle, USA (November 1998)
7. Hagen, C., Alonso, G.: Exception handling in workflow management systems. IEEE Transactions on Software Engineering 26(10), 943–958 (2000)
8. Casati, F., Fugini, M., Mirbel, I.: An environment for designing exceptions in workflows. Information Systems 24(3), 255–273 (1999)
9. Ackerman, M.S., Halverson, C.: Considering an organization's memory. In: Proceedings of the ACM 1998 Conference on Computer Supported Cooperative Work, pp. 39–48. ACM Press, Seattle, Washington, USA (1998)
10. Larkin, P.A.K., Gould, E.: Activity theory applied to the corporate memory loss problem. In: Svennson, L., Snis, U., Sorensen, C., Fagerlind, H., Lindroth, T., Magnusson, M., Ostlund, C. (eds.) Proceedings of IRIS 23 Laboratorium for Interaction Technology, University of Trollhattan Uddevalla, Sweden (2000)
11. Bardram, J.E.: I love the system - I just don't use it! In: Jakob, E. (ed.) GROUP 1997. Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, Phoenix, Arizona, USA, pp. 251–260. ACM Press, New York (1997)
12. Engestrom, Y., Miettinen, R., Punamaki, R.-L. (eds.): Perspectives on Activity Theory. Cambridge University Press, Cambridge (1999)
13. Nardi, B.A. (ed.): Context and Consciousness: Activity Theory and Human-Computer Interaction. MIT Press, Cambridge, Massachusetts (1996)
14. Adams, M., Edmond, D., ter Hofstede, A.H.M.: The application of activity theory to dynamic workflow adaptation issues. In: PACIS 2003. Proceedings of the 2003 Pacific Asia Conference on Information Systems, Adelaide, Australia, pp. 1836–1852 (July 2003)

15. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A service-oriented implementation of dynamic flexibility in workflows. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. LNCS, vol. 4275, pp. 291–308. Springer, Heidelberg (2006)
16. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Facilitating flexibility and dynamic exception handling in workflows through worklets. In: Bello, O., Eder, J., Pastor, O., Cunha, J.F. (eds.) CAiSE 2005 Forum, pp. 45–50, FEUP Edicoes, Porto (2005)
17. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Workflow exception patterns. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 288–302. Springer, Heidelberg (2006)
18. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems 30(4), 245–275 (2005)
19. van der Aalst, W.M.P., Aldred, L., Dumas, M., ter Hofstede, A.H.M.: Design and implementation of the YAWL system. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 142–159. Springer, Heidelberg (2004)
20. Hagen, C., Alonso, G.: Flexible exception handling in process support systems. Technical report No. 290, ETH Zurich, Switzerland (1998)
21. Lei, Y., Singh, M.P.: A comparison of workflow metamodels. In: Proceedings of the ER-97 Workshop on Behavioral Modeling and Design Transformations: Issues and Opportunities in Conceptual Modeling, Los Angeles, California, USA (November 1997)
22. Goodenough, J.B.: Exception handling: issues and a proposed notation. Communications of the ACM 18(12), 683–696 (1975)
23. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(3), 5–51 (2003)
24. Compton, P., Jansen, B.: Knowledge in context: A strategy for expert system maintenance. In: Barter, C.J., Brooks, M.J. (eds.) AI 1988. LNCS, vol. 406, pp. 292–306. Springer, Heidelberg (1990)
25. Scheffer, T.: Algebraic foundation and improved methods of induction of ripple down rules. In: Proceedings of the 2nd Pacific Rim Workshop on Knowledge Acquisition, Sydney, Australia, pp. 279–292 (1996)
26. Drake, B., Beydoun, G.: Predicate logic-based incremental knowledge acquisition. In: Compton, P., Hoffmann, A., Motoda, H., Yamaguchi, T. (eds.) Proceedings of the sixth Pacific International Knowledge Acquisition Workshop, Sydney, Australia, pp. 71–88 (December 2000)
27. Adams, M.: Facilitating Dynamic Flexibility and Exception Handling for Workflows. Phd thesis. Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia (2007), `http://yawlfoundation.org/documents/AdamsWorkletsFinalThesis.pdf`
28. zur Muehlen, M.: Workflow-based Process Controlling. Foundation, Design, and Implementation of Workflow-driven Process Information Systems. In: Advances in Information Systems and Management Science. vol. 6, Logos, Berlin (2004)
29. Casati, F., Pozzi, G.: Modelling exceptional behaviours in commercial workflow management systems. In: CoopIS 1999. Proceedings of the 4th IFCIS International Conference on Cooperative Information Systems, Edinburgh, Scotland, pp. 127–138. IEEE, Los Alamitos (1999)
30. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.-C.: Adaptive and dynamic composition in eFlow. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 13–31. Springer, Heidelberg (2000)

31. Muller, R., Greiner, U., Rahm, E.: AgentWork: a workflow system supporting rule-based workflow adaptation. Data & Knowledge Engineering 51(2), 223–256 (2004)
32. van Stiphout, R., Meijler, T.D., Aerts, A., Hammer, D., le Comte, R.: TREX: Workflow TRansactions by Means of EXceptions. Technical report, Eindhoven University of Technology (1997)
33. Dogac, A., Tambag, Y., Tumer, A., Ezbiderli, M., Tatbul, N., Hamali, N., Icdem, C., Beeri, C.: A workflow system through cooperating agents for control and document flow over the internet. In: Scheuermann, P., Etzion, O. (eds.) CoopIS 2000. LNCS, vol. 1901, pp. 138–143. Springer, Heidelberg (2000)
34. Weber, B., Wild, W., Breu, R.: CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In: Funk, P., González Calero, P.A. (eds.) ECCBR 2004. LNCS (LNAI), vol. 3155, pp. 434–448. Springer, Heidelberg (2004)
35. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes. In: Eder, J., Dustdar, S. (eds.) Business Process Management Workshops. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006)
36. Coleman, J.W.: Examining BPEL's compensation construct. In: REFT 2005. Proceedings of the Workshop on Rigorous Engineering of Fault-Tolerant Systems, Newcastle upon Tyne, UK, pp. 122–128 (July 2005)

Understanding the Occurrence of Errors in Process Models Based on Metrics

Jan Mendling[1], Gustaf Neumann[2], and Wil van der Aalst[3]

[1] BPM Cluster, Faculty of Information Technology
Queensland University of Technology, Australia
j.mendling@qut.edu.au
[2] Institute of Information Systems and New Media
Vienna University of Economics and Business Administration, Austria
neumann@wu-wien.ac.at
[3] Department of Computer Science,
Eindhoven University of Technology, The Netherlands
w.m.p.v.d.aalst@tue.nl

Abstract. Business process models play an important role for the management, design, and improvement of process organizations and process-aware information systems. Despite the extensive application of process modeling in practice, there are hardly empirical results available on quality aspects of process models. This paper aims to advance the understanding of this matter by analyzing the connection between formal errors (such as deadlocks) and a set of metrics that capture various structural and behavioral aspects of a process model. In particular, we discuss the theoretical connection between errors and metrics, and provide a comprehensive validation based on an extensive sample of EPC process models from practice. Furthermore, we investigate the capability of the metrics to predict errors in a second independent sample of models. The high explanatory power of the metrics has considerable consequences for the design of future modeling guidelines and modeling tools.

1 Introduction

Even though workflow and process modeling have been used extensively over the past 30 years, we know surprisingly little about the act of modeling and which factors contribute to a "good" process model in terms of error probability. This observation contrasts the large body of knowledge that is available for the formal analysis and verification of desirable properties, in particular for Petri nets. While conceptual work was conducted on guidelines and quality frameworks (e.g. [1,2,3,4]), there is clearly a need for an empirical research agenda to acquire new insights on quality (cf. [5]) and usage aspects (cf. [6]) of process modeling.

A recent study provides evidence that larger process models from practice tend to have more formal flaws (such as e.g. deadlocks) than smaller models [7,8]. One obvious hypothesis related to this phenomenon would be that human modelers loose track of the interrelations of large and complex models due to their limited cognitive capabilities (cf. [9]), and then introduce errors that they would not insert in a small model. Yet, there are further factors beyond simple count metrics such as the degrees of sequentiality, concurrency, or structuredness that need to be considered [10]. Against this background, this paper provides the following three contributions. First, we introduce a tool-based approach for detecting errors and calculating metrics for Event-driven Process Chains (EPCs), a popular business process modeling language. Second, we utilize an extensive sample of EPC models from practice to analyze the statistical connection between errors and metrics. Third, we calculate a logistic regression model and validate its ability to predict errors in a second independent sample. All these contributions relate to the formal correctness of the process model as a design artifact. Validation aspects with respect to the content of a process model, human understandability issues, ease of use of the modeling language, and modeling pragmatics are also closely related to quality, but they are not considered here.

The remainder of the paper is structured as follows. In Section 2 we give a brief overview of EPCs, EPC soundness, and the kind of metrics we calculate. In Section 3 we introduce a sample of 2003 EPCs from practice that we use to investigate the connection between errors and metrics. Moreover, we provide disaggregated descriptive statistics. In Section 4 we determine the correlation between errors and metrics, and estimate a logistic regression function. This function is validated against a second independent sample of EPCs for its capability to predict errors. Section 5 discusses our findings in the light of related research before Section 6 concludes the paper.

2 Error Detection and Metrics Calculation for EPCs

The Event-driven Process Chain (EPC) is a business process modeling language for representing temporal and logical dependencies of activities in a business process (see [11]). EPCs offer *function type* elements to capture activities of a process and *event type* elements describing pre- and post-conditions of functions. *Process interface type* elements are used to refer to subsequent processes. Furthermore, there are three kinds of *connector types* including AND (symbol $\wedge$), OR (symbol $\vee$), and XOR (symbol $\times$) for the definition of complex routing rules. Connectors have either multiple incoming and one outgoing arc (join connectors) or one incoming and multiple outgoing arcs (split connectors). The informal (or intended) semantics of an EPC can be described as follows. The AND-split activates all subsequent branches in concurrency. The XOR-split represents a choice between one of alternative branches. The OR-split triggers one, two or up to all of multiple branches based on conditions. In both cases of the XOR- and OR-split, the activation conditions are given in events subsequent to the connector. The AND-join waits for all incoming branches to complete, then it propagates control to the subsequent EPC element. The XOR-join merges alternative branches. The OR-join synchronizes all active incoming branches. This feature is called non-locality since the state of all transitive predecessor nodes has to be considered. Regarding their routing elements, EPCs are quite similar to BPMN [12] and YAWL [13]. Recently, EPC semantics have been formalized, and there is tool support for the verification of EPC soundness (see [14]). In this paper, we use EPC soundness as a correctness criterion in order to find out whether the model has errors or not. In particular, an EPC is sound if and only if for a set of initial markings I and a set of final markings O the following three properties hold:

(i) For each start-arc there exists an initial marking $i \in I$ where the arc (and hence the corresponding start event) holds a positive token.
(ii) For every marking reachable from an initial state $i \in I$, there exists a firing sequence leading from this marking to a final marking $o \in O$.
(iii) The final markings $o \in O$ are the only markings reachable from a marking $i \in I$ such that there is no node that can fire.

We use two complementary tools to check whether an EPC is sound (has no errors) or unsound (has errors): firstly, *xoEPC* that implements a fast, but not complete reduction rule approach, secondly, a ProM plug-in [15] that calculates the reachability graph which is complete, but not very performative [16]. Both tools can be coupled using the EPML interchange format [17].

Beyond verification of EPC soundness, *xoEPC* also calculates a set of process model metrics. We briefly describe them in the following list including their hypothetical connection with errors. The formulas for calculating the different metrics are given and extensively discussed in [16, Chap.5].[1] Furthermore, this reference mentions related work for each of the metrics.

Size. S_N refers to the number of nodes of the process model graph. An increase in S_N should imply an increase in error probability (+). Count metrics of different node types are written as e.g. S_C for connectors.

Diameter. *diam* gives the length of the longest path from a start node to an end node in the process model. It is presumably positively connected with error probability (+).

Density. Δ relates the number of arcs to the maximum number of arcs between all nodes. We presume a positive connection (+).

Coefficient of Connectivity. CNC gives the ratio of arcs to nodes (+).

Average Connector Degree. $\overline{d_C}$ gives the number of nodes a connector is in average connected to (+).

Maximum Connector Degree. $\widehat{d_C}$ captures the maximum degree over all connectors (+).

Separability. Π gives the ratio of the number of cut-vertices to the number of nodes. An increase in Π should imply a decrease in error probability (–).

Sequentiality. Ξ is the number of arcs between none-connector nodes divided by the overall number of arcs (–).

[1] This reference is also available online at http://wi.wu-wien.ac.at/home/mendling/publications/Mendling%20Doctoral%20thesis.pdf

Structuredness. Φ of the process graph is one minus the number of nodes in the EPC reduced with structured reduction rules divided by the number of nodes in the original EPC (–).

Depth. Λ captures how deep nodes are nested between splits and joins (+).

Connector Mismatch. MM gives the sum of mismatches for each connector type. The mismatch is the absolute sum of all input arcs minus output arcs over all connectors of a connector type (+).

Connector Heterogeneity. CH gives the type entropy of the connectors (+).

Control Flow Complexity. CFC sums up all choice of a process based on the number of splits of each type and its number of outgoing arcs (+).

Cyclicity. CYC relates nodes on cycles to all nodes (+).

Token Splits. TS sums up all concurrent threads that can be activated by AND- and OR-splits in the process (+).

Figure 1 illustrates for an example EPC taken from [18] which nodes and arcs contribute to the more elaborate metrics. Since the different count metrics, in particular for size, can be easily read from the model, we focus on those that need to be calculated from the process graph, i.e. separability, sequentiality, structuredness, depth, cyclicity, and diameter.

The *separability ratio* Π depends on the identification of cut vertices (i.e. articulation points), i.e. those nodes whose deletion breaks up the graph in two or more disconnected components. Figure 1 displays articulation points with a letter A written next to the top left-hand side of the node. For example, if the function "record loan request" is deleted, the start event is no longer connected with the rest of the process model. There are eleven articulation points in total yielding a separability ratio of $11/(27 - 2) = 0.440$. Note that start and end events do not belong to the set of articulation points, since their deletion does not increase the number of separate components.

The *sequentiality ratio* Ξ is calculated by relating the number of sequence arcs, i.e. arcs that connect functions and events, to the total number of arcs. Figure 1 highlights sequence arcs with an s label. There are ten sequence arcs and 29 arcs altogether which results in a sequentiality ratio of $10/29 = 0.345$. The degree of *structuredness* Φ relates the size of a reduced process model to the size of the original one. Figure 1 shows those elements with a cross on the left-hand side that are eliminated by reduction of trivial constructs. Other structured reduction rules are not applicable. Since 15 elements are deleted by reduction, the structuredness ratio is $1 - 15/27 = 0.556$. The *in-depth and out-depth* is also indicated for each node in Figure 1. The depth of a node is then the minimum of in-depth and out-depth. Several nodes have a depth of 1, which is a maximum, and therefore also the depth of the overall process. The *cyclicity* is based on the relation between number of nodes on a cycle and nodes in total. Figure 1 shows nodes on a cycle with a letter C written to the left-hand side bottom. There are seven such nodes yielding a cyclicity ratio of $7/27 = 0.259$. Finally, Figure 1 connects those 14 nodes that are on the *diameter* with a bold line.

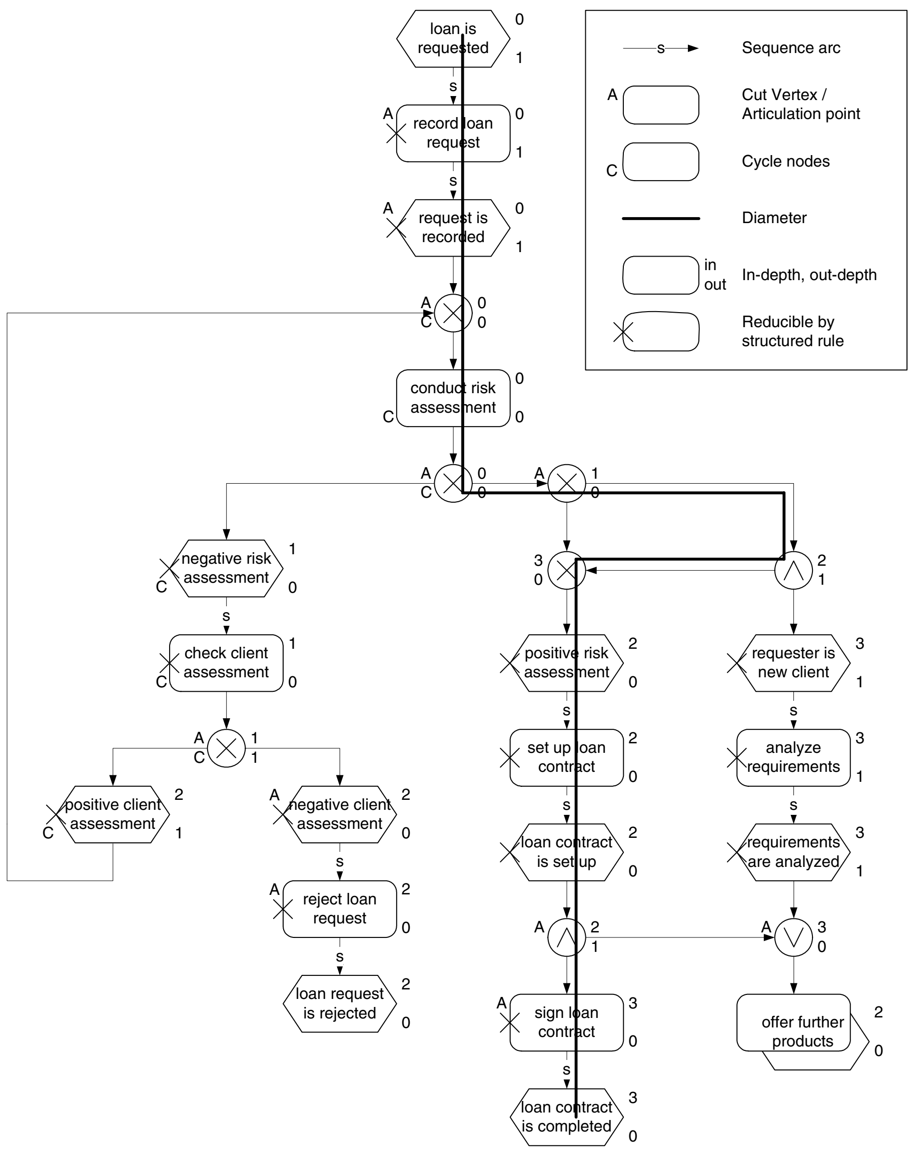


Fig. 1. EPC example with sequence arcs, articulation points, cycle nodes, diameter, depth, and reducible nodes

3 Empirical Distribution of Errors and Metrics

As input to our analysis we use a sample of EPC business process models that are available in the XML interchange format of ARIS Toolset of IDS Scheer

AG. The sample includes four collections of EPCs with a total of 2003 process models. All EPCs of the four groups were developed by practitioners.

1. *SAP Reference Model:* The first collection of EPCs is the SAP Reference Model. The development of the SAP reference model started in 1992 and first models were presented at CEBIT'93 [19, p.VII]. We use the SAP reference model in its version from 2000 that includes 604 non-trivial EPCs.
2. *Service Model:* The second collection of EPCs stems from a German process reengineering project in the service sector. The project was carried out in the late 1990s. The models of this project include 381 non-trivial EPCs.
3. *Finance Model:* The third model collection contains EPCs of a process documentation project in the Austrian financial industry. It includes 935 EPCs.
4. *Consulting Model:* The fourth collection covers in total 83 EPCs from three different consulting companies.

As a first step, the set of ARIS XML files is read and processed by *xoEPC* to generate information on errors and values for all the metrics. Furthermore, each EPC is then checked by the help of the reachability analysis plug-in for ProM. The results of this analysis are added to an analysis table. We use the software package for the statistical analysis of this table. In particular we present descriptive statistics disaggregated by group and error in Sections 3.1 and 3.2.

3.1 Descriptive Statistics Disaggregated by Group

In this section we characterize the overall EPC sample and its four sub-groups by the help of mean values μ and standard deviation σ for each metric. Several of the disaggregated mean values are quite close to each other, but in particular the Finance Model shows a striking differences: it has the highest mean in structuredness and sequentiality. Figures 2 and 3 illustrates the distribution

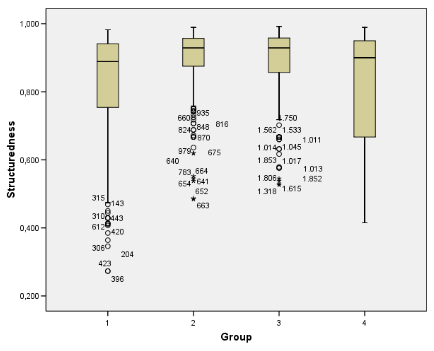


Fig. 2. Box plot for structuredness disaggregated by group (1=SAP, 2=Service, 3=Finance, and 4=Consulting)

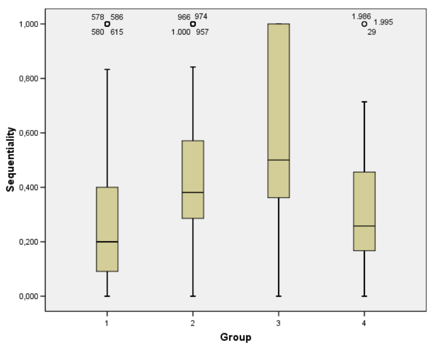


Fig. 3. Box plot for sequentiality disaggregated by group (1=SAP, 2=Service, 3=Finance, and 4=Consulting)

Table 1. Errors in the sample models

Parameter	Complete Sample	SAP Ref. Model	Services Model	Finance Model	Consulting Models
xoEPC errors	154	90	28	26	10
Unreduced EPCs	156	103	18	17	18
ProM error EPCs	115	75	16	7	17
EPCs with errors	215	126	37	31	21
EPCs in total	2003	604	381	935	83
Error ratio	10.7%	20.9%	9.7%	3.3%	25.3%

of both these metrics as box plots disaggregated by group. In this type of diagram invented by *Tukey* [20] the median is depicted as a horizontal line in a box that represents the interval between lower and upper quartile, i.e. the EPCs ranked by the metric from 25% to 75%. The upper and lower wicks define a one and a half multiple of the respective quartile. Values outside these two intervals are drawn as individual points and are considered to be outliers. From this observation on structuredness and sequentiality we might conclude that the Finance Model contains the more structured EPCs and thus might have less error models.

There is some evidence for such a hypothesis when we look at the number of errors in each of the four groups. Table 1 gives a respective overview. It can be seen that there are 2003 EPCs in the overall sample and 215 of them have at least one error. Accordingly, there is an overall error ratio of 10.7%. 154 of the 215 errors were found by *xoEPC*. 156 EPCs could not be reduced completely and were analyzed with ProM. This analysis revealed that 115 of the unreduced EPCs still had errors. Please note that there are EPCs for which both *xoEPC* and ProM found errors. Therefore, the number of EPCs with errors is less than the sum of EPCs with *xoEPC* and ProM errors. The comparison of the groups shows that the error ratio is quite different. In the previous paragraph we hypothesized that the finance model group might have less errors since its models are more structured. This suggests that metrics could be able to explain the low error ratio of only 3.3 %. We search further evidence in the next section.

3.2 Descriptive Statistics Disaggregated by hasErrors

In this section we discuss the distribution of the different metrics disaggregated by the variable *hasErrors*. Table 2 shows that there are quite large differences in the mean values of the sub-samples with and without errors. It is interesting to note that the error mean μ_e is higher than the non-error mean μ_n for most metrics where we assumed a positive connection with error probability in Section 2 and smaller for those metrics with a presumably negative connection. The only case where this does not hold is the density metric; it seems that it more accurately works as a counter-indicator for size than as an indicator for the density of connections in the model. The two columns on the right hand

Table 2. Mean and Standard Deviation of the sample models disaggregated by error

Parameter	Complete Sample		Non-Error EPCs		Error EPCs		2 σ dev. up	2 σ dev. down
	μ	σ	μ_n	σ_n	μ_e	σ_e	$\mu_n + 2\sigma_n$	$\mu_n - 2\sigma_n$
S_N	20.71	16.84	18.04	13.48	42.97	24.08	44.99 $\approx \mu_e$	
S_E	10.47	8.66	9.06	6.69	22.17	13.19	22.45 $\approx \mu_e$	
S_F	5.98	4.94	5.67	4.65	8.53	6.33	14.97	
S_C	4.27	5.01	3.30	3.47	12.26	7.89	10.24 $< \mu_e$	
S_A	21.11	18.87	18.14	15.20	45.79	26.78	48.54 $\approx \mu_e$	
$diam$	11.45	8.21	10.63	7.71	18.25	9.01	26.06	
Δ	0.09	0.07	0.09	0.07	0.03	0.02	0.23	
CNC	0.96	0.13	0.95	0.13	1.05	0.08	1.21	
$\overline{d_C}$	3.56	2.40	2.80	1.66	3.57	0.68	6.11	
$\widehat{d_C}$	2.88	1.60	3.31	2.28	5.64	2.41	7.87	
Sep. Π	0.56	0.27	0.59	0.27	0.35	0.13		0.06
Seq. Ξ	0.46	0.31	0.49	0.30	0.18	0.14		-0.12
Strct. Φ	0.88	0.11	0.90	0.09	0.70	0.16		0.72 $> \mu_e$
Depth Λ	0.70	0.74	0.61	0.69	1.45	0.73	1.98	
MM	3.31	4.55	2.54	3.45	9.71	6.92	9.44 $< \mu_e$	
CH	0.28	0.35	0.22	0.32	0.75	0.19	0.85	
CFC	382.62	8849.48	202.19	6306.23	1883.17	19950.26	12814.64	
CYC	0.01	0.08	0.01	0.06	0.07	0.17	0.12	
TS	1.82	3.53	1.28	2.46	6.26	6.62	6.20 $< \mu_e$	

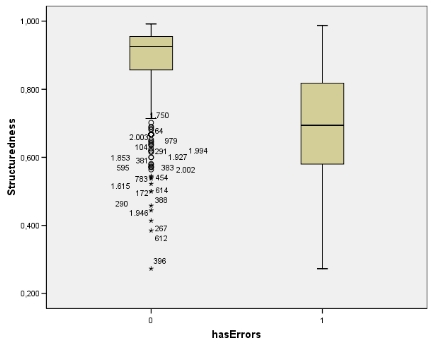


Fig. 4. Box plot for structuredness disaggregated by error

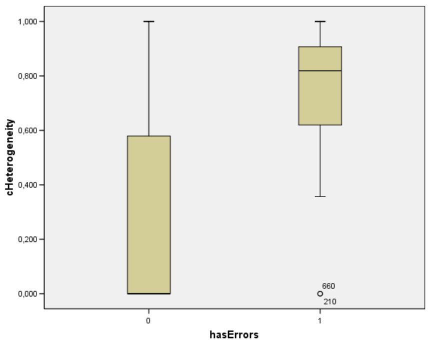


Fig. 5. Box plot for connector heterogeneity disaggregated by error

side of Table 2 might provide the basis for proposing potential error thresholds. The first of these columns gives a double σ_n deviation upwards from the non-error mean μ_n. Given a normal distribution only 2.5% of the population can be expected to have a metric value greater than this. The comparison of this value to the mean μ_e of the error EPCs gives an idea how good the two subparts of the sample can be separated by the metric. In several cases the mean μ_e is outside the double σ_n interval around μ_n. The box plots in Figures 4 and 5 illustrate the different distributions. It can be seen that correct EPCs tend to have much higher structuredness values and lower connector heterogeneity values. The next section investigates this observation with inferential statistics.

Table 3. Spearman correlation between hasError and metrics ordered by absolute correlation

	hasError		hasError
cHeterogeneity	0.46	Sequentiality	-0.35
C	0.43	Depth	0.34
MM	0.42	MaxCDegree	0.33
CFC	0.39	CYC	0.30
A	0.38	diameter	0.30
tokenSplit	0.38	Separability	-0.29
N	0.38	CNC	0.28
E	0.38	AvCDegree	0.23
Density	-0.37	F	0.19
Structuredness	-0.36		

4 Inferential Statistics

4.1 Correlation Analysis

This section approaches the connection between error probability and metrics with a correlation analysis. We use the Spearman rank correlation coefficient for ordinal data. As a confirmation of the previous observation all variables have the expected direction of influence besides the density metric. Table 3 presents the Spearman correlation between *hasErrors* and the metrics ordered by strength of correlation. It can be seen that several correlations are quite considerable with absolute values between 0.30 and 0.50. The significance of all correlations is good with 99% confidence.

The ability of a metric to separate error from non-error models by ranking is illustrated in Figures 6 and 7. For Figure 6 all models are ranked according to their size. A point (x, y) in the graph relates a size x to the relative frequency of error models in a subset of models that have at least size x, i.e. $y = |\{\frac{|errorEPCs|}{|allEPCs|} \mid S_N(EPC) > x\}|$. It can be seen that the relative frequency of error EPCs increases by increasing the minimum number of nodes. In particular, the relative frequency of error EPCs is higher than 50% for all EPCs of at least 48 nodes. In Figure 6 all models are ranked according to their structuredness and (x, y) relates the structuredness x to the subset of models that have

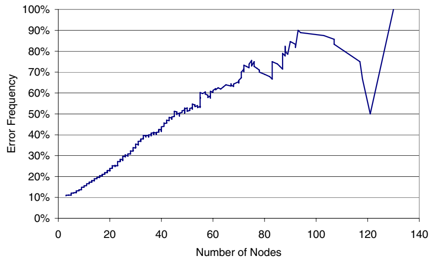


Fig. 6. Error frequency to ordered number of nodes

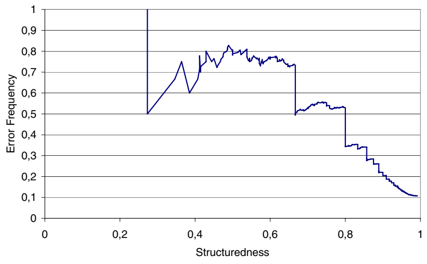


Fig. 7. Error frequency to ordered structuredness

at most structuredness x. Here, the graph decreases and drops below 50% at a structuredness value of 0.80. Similar observations can be made for some of the other metrics, too. The values could be used as candidate thresholds. Altogether the relative frequency of error models above 50% is reached if

number of nodes $S_N > 48$	number of arcs $S_A > 62$
number of connectors $S_C > 8$	token splits $TS > 7$
number of events $S_E > 22$	connector mismatch $MM > 9$
number of functions $S_F > 40$	structuredness $\Phi < 0.8$

4.2 Logistic Regression Estimation

This section provides an introduction to logistic regression analysis and presents the result of its application for estimating the prediction model for error probability based on metrics. Logistic regression is a statistical model to estimate the probability of binary choices. It is perfectly suited to deal with dependent variables such as *hasErrors* with its range *error* and *no error*. The idea of binary choice models is to describe the probability of a binary event by its odds, i.e., the ratio of event probability divided by non-event probability. In the *logistic regression* (or *logit*) model the odds are defined as $logit(p_i) = ln(\frac{p_i}{1-p_i}) = \beta_0 + \beta_1 x_{1,i} + \ldots + \beta_k x_{k,i}$ for k input variables and i observations, i.e. i EPCs in our context. From this follows that

$$p_i = \frac{e^{\beta_0 + \beta_1 x_{1,i} + \ldots + \beta_k x_{k,i}}}{1 + e^{\beta_0 + \beta_1 x_{1,i} + \ldots + \beta_k x_{k,i}}}$$

The relationship between input and dependent variables is represented by an S-shaped curve of the logistic function that converges to 0 for $-\infty$ and to 1 for ∞. The cut value of 0.5 defines whether event or non-event is predicted. $Exp(B)$ gives the multiplicative change of the odds if the input variable is increased by one unit, i.e. $Exp(B) > 1$ increases and $Exp(B) < 1$ decreases error probability. The actual value $Exp(B)$ cannot be interpreted in isolation since its impact depends upon the position on the non-linear curve [21, p.791]. The significance of a logistic regression model is assessed by the help of two statistics. First, the *Hosmer & Lemeshow Test* should be greater than 5% to indicate a good fit based on the difference between observed and predicted frequencies. Second, *Nagelkerke's* R^2 ranging from 0 to 1 serves as a coefficient of determination indicating which fraction of the variability is explained. Furthermore, each estimated coefficient of the logit model is tested using the *Wald statistic* for being significantly different from zero. The significance should be less than 5%. We calculate the logistic regression model based on a stepwise introduction of those variables that provide the greatest increase in likelihood. For more on logistic regression see [22].

Before calculating a multivariate logistic regression model for error probability we carry out two preparatory analyses. First, we check collinearity, then we determine which variables are included in the regression model. Furthermore, we excluded 29 EPCs from the analysis that were not syntactically correct.

Collinearity describes the phenomenon that at least one of the independent variables can be represented as a linear combination of other variables. The absence of collinearity is not a hard criterion for the applicability of logistic regression, but it is desirable. Since the tolerance test indicated collinearity problems, we had to dropped all count metrics apart from S_N since they were highly correlated. This resulted in a reduced variable set with no collinearity problems. Furthermore, we calculated *univariate models* with and without a constant in order to check whether all inputs were significantly different from zero. As a conclusion from this analysis we drop the constant and the control flow complexity CFC for the multivariate analysis. First, the constant is not significantly different from zero (Wald statistic of 0.872 and 0.117) in the separability and the sequentiality model which suggests that it is not necessary. Second, the CFC metric is not significantly different from zero (Wald statistic of 0.531 and 0.382) in both models with and without constant. All other metrics stay in the set of input variables from the multivariate logistic regression model.

The final model was calculated in nine steps and it includes seven variables. It is interesting to note that again the hypothetical impact direction of the included metrics is confirmed. All variables have an excellent *Wald statistic* value of better than 0.001 indicating that they are significantly different from zero. Furthermore, the *Hosmer & Lemeshow test* is greater than 0.05 which is also a good value. Finally, the *Nagelkerke* R^2 has an excellent value of 0.901 indicating a high degree of explanation. Based on the regression results we can derive a classification function $p(EPC)$ for EPCs. It predicts that the EPC has errors if the result is greater than 0.5. Otherwise it predicts that there are no errors in the EPC. It is calculated by the help of the metrics coefficient of connectivity CNC, connector mismatch MM, cyclicity CYC, separability Π, structuredness Φ, connector heterogeneity CH, and the diameter. It is

$$p(EPC) = \frac{e^{logit(EPC)}}{1 + e^{logit(EPC)}}$$

with

$$\begin{aligned} logit(EPC) = &+4.008\ CNC \\ &+0.094\ MM \\ &+3.409\ CYC \\ &-2.338\ \Pi \\ &-9.957\ \Phi \\ &+3.003\ CH \\ &+0.064\ diameter \end{aligned}$$

It is easy to calculate an error prediction for an EPC based on this function. For the sample this function yields the following classification:

– 1724 EPCs are correctly predicted to have no errors,
– 155 EPCs are correctly predicted to have errors,
– 58 EPCs are predicted to have no errors, but actually had, and
– 37 EPCs are predicted to have errors, but actually had none.

Altogether 1879 EPCs have the correct prediction. The overall percentage is 95.2%, that is 6% better than the naive model that always predicts no error (89.2%). Furthermore, there are 213 EPCs with errors in the reduced sample. 155 of them are correctly predicted, i.e. 72.7%. Finally, the prediction function gives a clue about the relative importance of the different metrics. Structuredness appears to be the most important parameter since its absolute value is three times as high as the second. Then, the coefficient of connectivity, cyclicity, separability, and connector heterogeneity seem to be of comparable importance. Finally, connector mismatch and the diameter might be of minor importance.

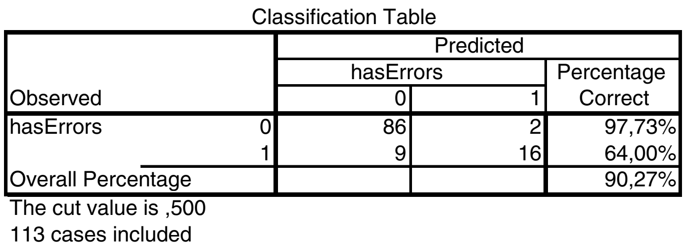
In the following section we analyze how good the regression function is able to forecast errors in a sample of EPCs that was not included in the estimation.

4.3 Logistic Regression Validation

In this section we utilize the estimated function to predict errors in EPCs from a holdout sample. This step is of paramount importance for establishing the criterion validity of the measurements, i.e. their pragmatic value (cf. e.g. [23]). For testing the performance of the prediction function we gathered a sample from popular German EPC business process modeling textbooks. The sample includes 112 models from the following books in alphabetical order:

- *Becker and Schütte* [24]. This book discusses information systems in the retail sector with a special focus on conceptual modeling. In particular it covers 65 EPC models that we include in the holdout sample.
- *Scheer* [25]. This textbook is an introduction to the ARIS framework and uses reference models for production companies to illustrate it. From this book we includes 27 EPC reference models in the holdout sample.
- *Seidlmeier* [26]. This book is an introduction to the ARIS framework. It includes 10 EPCs that we include in the holdout sample.
- *Staud* [27]. This book focuses on business process modeling and in particular EPCs. We included 13 EPCs from this book in the holdout sample.

All EPCs of the holdout sample were checked for errors first with *xoEPC* and afterwards with the ProM plug-in. Altogether there are 25 of the 113 models that have errors, i.e. 21.43%. Based on the metrics generated by *xoEPC* we can


Classification Table

		Predicted		
		hasErrors		Percentage Correct
Observed		0	1	
hasErrors	0	86	2	97,73%
	1	9	16	64,00%
Overall Percentage				90,27%

The cut value is ,500
113 cases included

Fig. 8. Classification table for EPCs from the holdout sample

easily apply the prediction function. The result of this calculation is summarized in the classification table in Figure 8. It can be seen that 102 of the 113 EPCs are classified correctly, i.e. 86 models without errors are predicted to have none and 16 with errors are predicted to have at least one. Altogether 90.27% of the 113 EPCs were predicted correctly. Please note that there is a difference in the interpretation of this classification result and the one in Section 4.2. During the estimation of the logistic regression the sample is known and used to tune the coefficients. Here, we use this function to classify an independent sample. Based on the De Moivre-Laplace theorem, we are able to calculate a confidence interval for the accuracy of the prediction function. With a confidence value of 95% it yields an accuracy interval from 81.15% to 96.77%, i.e. the prediction can be expected to be correct in at least 81.15% of the cases with a 95% confidence. This result strongly supports the validity of the function for predicting error probability.

4.4 Implications of the Findings

In this section we have conducted several statistical analyses related to the hypotheses on a connection between metrics and error probability. The results strongly confirm the hypotheses since the mean difference between error and non-error models, the correlation coefficients, and the regression coefficients confirm the hypothetical impact direction of all metrics except the density metric (see Table 4). This metric appears to be more closely related to the inverse of size than the relative number of arcs of an EPC.

Table 4. Hypothetical and empirical connection between metrics and errors

	Hypothetical connection	$\mu_e - \mu_n$	Correlation	Regression coefficient	Direction
S_N	+	24.93	0.38		confirmed
S_E	+	13.11	0.38		confirmed
S_F	+	2.86	0.19		confirmed
S_C	+	8.96	0.43		confirmed
S_A	+	27.64	0.38		confirmed
$diam$	+	7.62	0.30	0.064	confirmed
Δ	+	-0.06	-0.37		not confirmed
CNC	+	0.11	0.28	4.008	confirmed
$\overline{d_C}$	+	0.76	0.23		confirmed
$\widehat{d_C}$	+	2.33	0.33		confirmed
Sep. Π	-	-0.24	-0.29	-2.338	confirmed
Seq. Ξ	-	-0.31	-0.35		confirmed
Strct. Φ	-	-0.20	-0.36	-9.957	confirmed
Depth Λ	+	0.85	0.34		confirmed
MM	+	7.18	0.42	0.094	confirmed
CH	+	0.54	0.46	3.003	confirmed
CFC	+	1680.99	0.39		confirmed
CYC	+	0.06	0.30	3.409	confirmed
TS	+	4.97	0.38		confirmed

These results have strong implications for the quality of business process modeling:

1. The connection of the metrics with error probability provides a theoretical and empirical basis for defining process modeling principles and guidelines. The analysis reveals that in particular structured models are less error prone.
2. The established connection builds a foundation for a measurement-based management approach for the process of business process modeling. Different design alternatives can be discussed more objectively on the metric values.
3. The design of future business process modeling tools can benefit from these findings by providing online feedback to the modeler when a certain metric passes an error threshold.
4. It has also some implications on the level of the process modeling language. Considering that the connector heterogeneity has an impact on error probability it might be a good idea to restrict modeling to the two connector types AND and XOR, and use OR-connectors only in structured blocks. Furthermore, there was a strong correlation between the number of start and end events with error probability. This fact suggests to restrict the use of multiple starts and ends. Modelers seem to loose track of the allowed combinations of these elements quite fast. In the reduced set of EPCs there are several EPCs for which no combination of start events guarantees a proper execution.
5. The results have implications for the teaching of business process modeling. On the one hand, the large number of errors suggests that practitioners frequently have problems to understand the behavioral implications of their design. On the other hand, the metrics are a good starting point to teach patterns that are unlikely to result in errors.

5 Related Work

There are basically two main streams of research related to our work in the conceptual modeling area: top-down quality frameworks and bottom-up metrics that relate to quality aspects. For related work on Petri net verification refer to [28] and on EPCs to [16].

One prominent *top-down quality framework* is the SEQUAL framework [1,4]. It builds on semiotic theory and defines several quality aspects based on relationships between a model, a body of knowledge, a domain, a modeling language, and the activities of learning, taking action, and modeling. Its usefulness was confirmed in an experiment [29]. The Guidelines of Modeling (GoM) [2] define an alternative quality framework that is inspired by general accounting principles. The guidelines include the six principles of correctness, clarity, relevance, comparability, economic efficiency, and systematic design. This framework was operationalized for EPCs and also tested in experiments [2]. Furthermore, there are authors (e.g. [5]) advocating a specification of a quality framework for conceptual modeling in compliance with the ISO 9126 standard [30] for software quality. A respective adaptation to business process modeling is reported in [31]. Our research complements these approaches regarding semantical correctness. While the frameworks tend to be rather abstract, we find strong support for operational recommendations like using structured building blocks and limiting the number of nodes in a single process model.

Much work has been done related to *bottom-up metrics that relate to quality aspects* of process models, stemming from different research and partially isolated from each other (see [32,33,34,35,36,37,38,39,40,10] or for an overview [16]). Several of these contributions are theoretic without empirical validation. Most authors doing experiments focus on the relationship between metrics and quality aspects: *Canfora et al.* study the connection between mainly count metrics for e.g. activities or routing elements and maintainability of software process models [38]; *Cardoso* validates the correlation between control flow complexity and perceived complexity [41]; and *Mendling et al.* use metrics to predict control flow errors such as deadlocks in process models [8,10]. The results of this research confirm the negative connection between size and quality aspects. Beyond that, it extends this stream of research with a validation of an error prediction function that was derived by the help of an extensive sample of process models from practice.

Finally, there are some further surveys that investigate the maturity [42], usability [43], and understandability of business process modeling languages [44]. They also relate to quality aspects of process models, but not directly to the connection of errors and metrics.

6 Conclusions and Future Work

With this paper, we addressed the shortage of empirical insight into business process modeling and its quality parameters in practice. In particular, we used a collection of 2003 EPC business process models from practice, and determined for each of the models whether they have errors or not. Furthermore, we calculated an extensive set of metrics for each model. Based on this data, we were able to show that several metrics have a strong statistical connection with the occurrence of errors, and that most of the metrics increase or decrease error probability as expected. Using a logistic regression model, we could even derive a prediction function that accurately classifies models as having errors or not based on metrics.

These findings clearly demonstrate that errors do not occur by chance in business process models, and that certain characteristics like structuredness are desirable for avoiding errors. In future research we aim to conduct further experiments to test the connection between the metrics and other quality aspects of modeling like understandability and maintainability. As a result from these experiments, we expect to define new business process modeling guidelines which are metrics-based, which can be easily translated into operations, and which lead to high quality business process models.

References

1. Lindland, O.I., Sindre, G., Sølvberg, A.: Understanding quality in conceptual modeling. IEEE Software 11, 42–49 (1994)
2. Becker, J., Rosemann, M., Uthmann, C.: Guidelines of Business Process Modeling. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. Models, Techniques, and Empirical Studies, pp. 30–49. Springer, Berlin (2000)
3. Hoppenbrouwers, S.S., Proper, H.E., van der Weide, T.: A Fundamental View on the Process of Conceptual Modeling. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 128–143. Springer, Heidelberg (2005)
4. Krogstie, J., Sindre, G., Jørgensen, H.: Process models representing knowledge for action: a revised quality framework. European Journal of Information Systems 15, 91–102 (2006)
5. Moody, D.: Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. Data & Knowledge Engineering 55, 243–276 (2005)
6. Davies, I., Green, P., Rosemann, M., Indulska, M., Gallo, S.: How do practitioners use conceptual modeling in practice? Data & Knowledge Engineering 58, 358–380 (2006)
7. Mendling, J., Moser, M., Neumann, G., Verbeek, H., Dongen, B., van der Aalst, W.: Faulty EPCs in the SAP Reference Model. In: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, pp. 451–457. Springer, Heidelberg (2006)
8. Mendling, J., Moser, M., Neumann, G., Verbeek, H., Dongen, B., van der Aalst, W.: Detection and Prediction of Errors in EPCs of the SAP Reference Model. Data & Knowledge Engineering (accepted for publication)
9. Simon, H.: Sciences of the Artificial, 3rd edn. The MIT Press, Cambridge (1996)
10. Mendling, J., Neumann, G.: Error metrics for business process models. Technical Report JM-2006-12-03, Vienna Univ. of Econ. and Business Administration (2006)
11. Keller, G., Nüttgens, M., Scheer, A.W.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany (1992)
12. OMG, (ed.): Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification, dtc/06-02-01, Object Management Group (2006)
13. van der Aalst, W., ter Hofstede, A.: YAWL: Yet Another Workflow Language. Information Systems 30, 245–275 (2005)
14. Mendling, J., van der Aalst, W.: Formalization and Verification of EPCs with OR-Joins Based on State and Context. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007. LNCS, vol. 4495, Springer, Heidelberg (2007)
15. van Dongen, B., Medeiros, A., Verbeek, H., Weijters, A., van der Aalst, W.: The ProM framework: A New Era in Process Mining Tool Support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)
16. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD thesis, Vienna University of Economics and Business Administration (2007)
17. Mendling, J., Nüttgens, M.: EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). Information Systems and e-Business Management 4, 245–263 (2006)

18. Nüttgens, M., Rump, F.J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: Desel, J., Weske, M. (eds.) Proceedings of Promise 2002, Potsdam, Germany. Lecture Notes in Informatics, vol. 21, pp. 64–77 (2002)
19. Keller, G., Teufel, T.: SAP(R) R/3 Process Oriented Implementation: Iterative Process Prototyping. Addison-Wesley, Reading (1998)
20. Tukey, J.: Exploratory Data Analysis. Addison-Wesley, Reading (1977)
21. Judge, G., Hill, R., Griffiths, W., Lütkepohl, H., Lee, T.C.: Introduction to the theory and practice of econometrics, 2nd edn. John Wiley & Sons, England (1988)
22. Hosmer, D., Lemeshow, S.: Applied Logistic Regression. Wiley & Sons, England (2000)
23. Marczyk, G., DeMatteo, D., Festinger, D.: Essentials of Research Design and Methodology. Wiley & Sons, Inc., England (2005)
24. Becker, J., Schütte, R.: Handelsinformationssysteme. Moderne Industrie (2004)
25. Scheer, A.-W.: Wirtschaftsinformatik: Referenzmodelle für industrielle Geschäftsprozesse. Springer, Heidelberg (1998)
26. Seidlmeier, H.: Prozessmodellierung mit ARIS. Vieweg Verlag (2002)
27. Staud, J.: Geschäftsprozessanalyse: Ereignisgesteuerte Prozessketten und Objektorientierte Geschäftsprozessmodellierung. Springer, Heidelberg (2006)
28. Reisig, W., Rozenberg, G. (eds.): Lectures on Petri Nets I: Basic Models. LNCS, vol. 1491. Springer, Heidelberg (1998)
29. Moody, D., Sindre, G., Brasethvik, T., Sølvberg, A.: Evaluating the quality of process models: Empirical testing of a quality framework. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 380–396. Springer, Heidelberg (2002)
30. ISO: Information technology - software product evaluation - quality characteristics and guide lines for their use. Iso/iec is 9126 (1991)
31. Gücegliogl u, A.S., Demirörs, O.: Using software quality characteristics to measure business process quality. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 374–379. Springer, Heidelberg (2005)
32. Lee, G., Yoon, J.M.: An empirical study on the complexity metrics of petri nets. Microelectronics and Reliability 32, 323–329 (1992)
33. Nissen, M.: Redesigning reengineering through measurement-driven inference. MIS Quarterly 22, 509–534 (1998)
34. Morasca, S.: Measuring attributes of concurrent software specifications in petri nets. In: METRICS 1999. Proceedings of the 6th International Symposium on Software Metrics, pp. 100–110. IEEE Computer Society Press, Washington, DC, USA (1999)
35. Reijers, H., Vanderfeesten, I.: Cohesion and coupling metrics for workflow process design. In: Desel, J., Pernici, B., Weske, M. (eds.) BPM 2004. LNCS, vol. 3080, pp. 290–305. Springer, Heidelberg (2004)
36. Cardoso, J.: Evaluating Workflows and Web Process Complexity. In: Workflow Handbook 2005, Future Strategies Inc., 284–290 (2005)
37. Balasubramanian, S., Gupta, M.: Structural metrics for goal based business process design and evaluation. Business Process Management Journal 11, 680–694 (2005)
38. Canfora, G., García, F., Piattini, M., Ruiz, F., Visaggio, C.: A family of experiments to validate metrics for software process models. Journal of Systems and Software 77, 113–129 (2005)

39. Aguilar, E.R., Ruiz, F., García, F., Piattini, M.: Towards a Suite of Metrics for Business Process Models in BPMN. In: Manolopoulos, Y., Filipe, J., Constantopoulos, P., Cordeiro, J. (eds.) ICEIS 2006. Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration (III), Paphos, Cyprus, May 23-27, 2006, pp. 440–443 (2006)
40. Laue, R., Gruhn, V.: Complexity metrics for business process models. In: Abramowicz, W., Mayr, H.C. (eds.) BIS 2006. 9th International Conference on Business Information Systems. Lecture Notes in Informatics, vol. 85, pp. 1–12 (2006)
41. Cardoso, J.: Process control-flow complexity metric: An empirical validation. In: IEEE SCC 2006. Proceedings of IEEE International Conference on Services Computing, Chicago, USA, September 18-22, pp. 167–173. IEEE Computer Society Press, Los Alamitos (2006)
42. Rosemann, M., Recker, J., Indulska, M., Green, P.: A study of the evolution of the representational capabilities of process modeling grammars. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 447–461. Springer, Heidelberg (2006)
43. Agarwal, R., Sinha, A.P.: Object-oriented modeling with UML: a study of developers' perceptions. Commun. ACM 46, 248–256 (2003)
44. Sarshar, K., Loos, P.: Comparing the control-flow of epc and petri net from the end-user perspective. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 434–439. Springer, Heidelberg (2005)

Data-Driven Modeling and Coordination of Large Process Structures[*]

Dominic Müller[1,2], Manfred Reichert[1], and Joachim Herbst[2]

[1] Information Systems Group, University of Twente, The Netherlands
{d.mueller, m.u.reichert}@ewi.utwente.nl
[2] Dept. GR/EPD, DaimlerChrysler AG
Group Research & Advanced Engineering, Germany
joachim.j.herbst@daimlerchrysler.com

Abstract. In the engineering domain, the development of complex products (e.g., cars) necessitates the coordination of thousands of (sub-) processes. One of the biggest challenges for process management systems is to support the modeling, monitoring and maintenance of the many interdependencies between these sub-processes. The resulting process structures are large and can be characterized by a strong relationship with the assembly of the product; i.e., the sub-processes to be coordinated can be related to the different product components. So far, sub-process coordination has been mainly accomplished manually, resulting in high efforts and inconsistencies. IT support is required to utilize the information about the product and its structure for deriving, coordinating and maintaining such *data-driven process structures*. In this paper, we introduce the COREPRO framework for the data-driven modeling of large process structures. The approach reduces modeling efforts significantly and provides mechanisms for maintaining data-driven process structures.

1 Introduction

Enterprises are increasingly demanding IT support for their business processes. One challenge emerging in this context is to coordinate the execution of large and long-running processes (e.g., related to car development). Engineering processes, for instance, often consist of numerous concurrently executed, interdependent sub-processes. The reasons for this fragmentation are manifold: Typically, these sub-processes are related to different (data) objects (e.g., product components), enacted by different organizational units (e.g., dealing with the *testing* or *releasing* of single components), and controlled by different IT systems. We denote such correlated sub-processes as *process structure*.

These process structures have in common that changes (e.g., removing a sub-process or adding a dependency between sub-processes) as well as real-world exceptions (e.g., abnormal termination of a sub-process) occur frequently and

[*] This work has been funded by *Daimler AG Group Research* and has been conducted in the COREPRO (COnfiguration based RElease PROcesses) project.

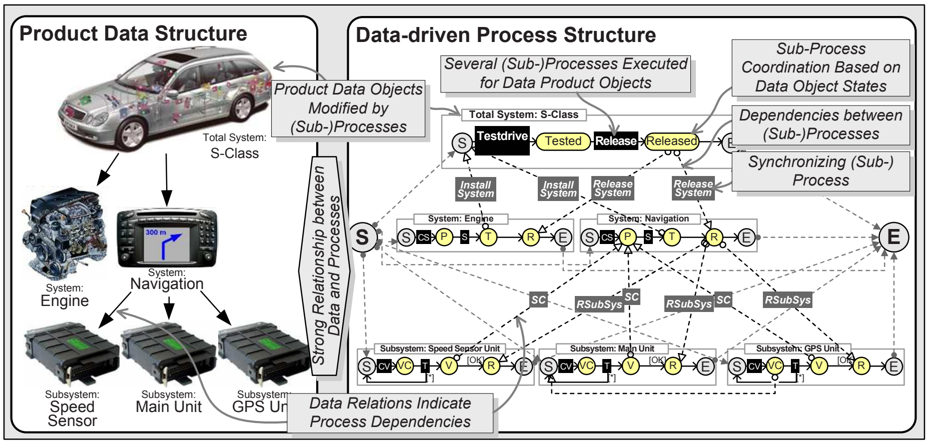

Product Data Structure
Total System: S-Class
System: Engine
System: Navigation
Subsystem: Speed Sensor
Subsystem: Main Unit
Subsystem: GPS Unit
Product Data Objects Modified by (Sub-)Processes
Strong Relationship between Data and Processes
Data Relations Indicate Process Dependencies
Data-driven Process Structure
Several (Sub-)Processes Executed for Data Product Objects
Sub-Process Coordination Based on Data Object States
Dependencies between (Sub-)Processes
Synchronizing (Sub-) Process
Testdrive
Tested
Release
Released
Install System
Release System
SC
RSubSys

Fig. 1. Example for a Data Structure and a Related Data-driven Process Structure

may affect not only single sub-processes but also the whole process structure [1,2]. Consequently, IT support must flexibly cover modeling, enactment and maintenance of process structures and assure their consistency. Even the modeling of process structures constitutes a challenging task since these structures usually comprise hundreds up to thousands of sub-processes (and sub-process dependencies). Doing this manually often results in errors or inconsistencies leading to bad process performance and high process costs.

To cope with these challenges, we have to better understand the dependencies between sub-processes. Case studies we conducted in the automotive industry [1,3] have revealed that the dependencies between the different sub-processes of a process structure typically base on the assembly of the product to be manufactured. As an example for a *product structure* (or configuration structure) consider the total electrical system in a modern car which consists of up to 300 interconnected components. To verify the functionality of the total system, several sub-processes (e.g., *testing* and *release*) have to be executed for each electrical (sub-)component. Interestingly, the technical relations between the different product components indicate sub-process dependencies; i.e., the relation between two components leads to dependencies between sub-processes modifying these components. We use the notion of *data-driven process structures* to characterize process structures, which are prescribed by respective data structures. Fig. 1 presents an example for a data-driven process structure. The strong relationship between data structures and process structures (e.g., the relations between the S-Class and the Navigation object leads to respective sub-process dependencies) also implies that a changed data structure (e.g., total electrical system for another car series without navigation) leads to a different process structure.

Our goal is to reduce modeling efforts for data-driven process structures by increasing model reusability and maintainability. In the automotive domain, for instance, we can benefit from the upcoming standardization of development processes driven by quality frameworks like CMMI (Capability Maturity Model Integration) or engineering guidelines (e.g., [4]). This leads to standardized processing of objects (e.g., the testing process for the speed sensor is independent from the car series it is built in), which can be utilized to increase reuse of process models and to reduce modeling efforts. In order to benefit from the standardization, a loose coupling of data structures and process structures is required. In particular, three issues arise:

1. How to describe the processing of single objects, i.e., the relationship between an object and its modifying sub-processes?
2. How to describe the processing of the overall data structure, i.e., the dependencies between the sub-processes in relation to the different data objects?
3. How to automatically derive a proper process structure that can be *customized* by the underlying data structure, i.e., different data structures lead to different process structures?

So far, there exists no IT support for configuring a process structure based on a given data structure. IT systems used in industry, such as product data management systems or workflow management tools currently lack an integration of data and processes [1]. Approaches from academia, such as data-centered process paradigms [5,6,7] also do not fully address the aforementioned issues. Instead, users have to manually define the requested process structure for each given data structure. This often leads to inflexible process structures and generates high efforts for coordinating and maintaining them. In this paper, we introduce the modeling component of the COREPRO approach, which aims at an intuitive and product-related integration of data and (sub-)processes. In particular, COREPRO enables

– the data-driven specification of process structures at the model level
– the automated creation of process structures based on given data structures
– the data-driven adaptation of process structures to deal with real-world changes.

We utilize the life cycles of objects (i.e., the sequence of states an object goes through during its lifetime) for enabling data-driven modeling and coordination of process structures. State transitions within a life cycle take place when a sub-process is enacted for the related object [8,9,10,11]. According to the relations between objects, we connect the life cycles of these objects. The concept thereby enables the automated derivation of the process structure for a given data structure.

The remainder of this paper is structured as follows: Section 2 characterizes the relationship between data and process structures, and introduces our approach for describing them. Section 3 shows how to model and instantiate (product) data structures. Section 4 deals with the data-driven creation and change of large process structures. Section 5 illustrates the practical benefits of the COREPRO approach. Section 6 discusses related work, and Section 7 concludes with a summary and outlook.

2 Overview of the Approach

IT support for the modeling and change of data-driven process structures must meet four major requirements. First, it must enable the definition of the (product) data structure, i.e., its objects and their relations. Second, with each (data) object a set of sub-processes for processing this object and for transforming its state has to be associated. Third, sub-process dependencies have to be defined based on the object relations. For example, a sub-process for the Navigation object shall be not started before having finished the sub-processes of the related subsystems (cf. Fig. 1). Fourth, the concepts must enable the automated creation of a data-driven process structure.

The COREPRO modeling framework meets these requirements. In order to enable reuse and to reduce modeling efforts, COREPRO distinguishes between the model and the instance level when creating data-driven process structures (cf. Fig. 2). We allow defining a domain specific *data model* consisting of object and relation types (cf. Step 1a in Fig. 2). Such a data model can then be instantiated to create specific *data structures* (e.g., a *bill of material*) consisting of objects and relations (cf. Step 1b in Fig. 2). While the definition of a data model requires profound domain knowledge, the instantiation can be done by users.

Further, process experts describe the dynamic aspects of each object type by modeling *object life cycles* (OLC). An OLC defines the coordination of sub-processes associated with a particular object type (cf. Step 2a in Fig. 2). A sub-process is an autonomous process (or activity). In COREPRO an OLC is mapped

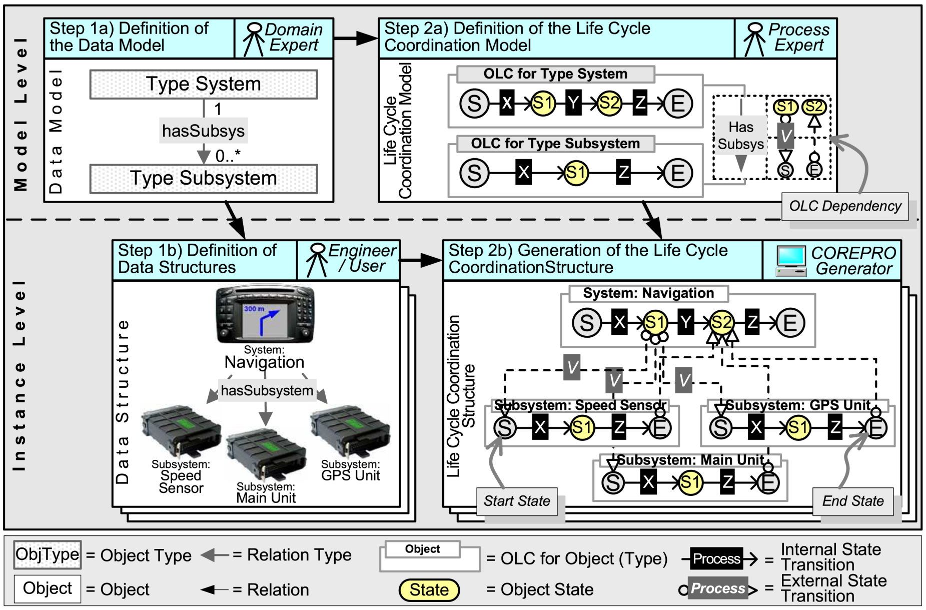


Fig. 2. Overall Concept of the COREPRO Modeling Approach

to a state transition system whose states correspond to object states and whose (internal) state transitions are triggered when associated sub-processes (which are modifying the object) are completed. In Fig. 2, the OLC for object type `System` (Step 2a), for example, goes through state `S1` followed by state `S2`. The *internal state transition* from `S1` to `S2` takes place when finishing sub-process `Y`. Altogether, an OLC constitutes an integrated view on a particular object and on the sub-processes manipulating this object.

Defining the dynamic aspects of each object type by modeling its OLC is only half of the story. We also have to deal with the many dependencies existing between the sub-processes associated with different objects types. Such dependencies might describe, for example, that every system (`Engine` and `Navigation`) must have reached a certain state before the `Testdrive` sub-process for the `S-Class` can be started (cf. Fig. 1). Consequently, a sub-process dependency can be seen as a synchronization link between the states of concurrently enacted OLCs.

In COREPRO, we specify such sub-process dependencies by defining *external state transitions*, which connect states of different OLCs. Like an internal state transition within an OLC, an external state transition can lead to the enactment of a sub-process. In Fig. 2, for example, the external state transition between the OLCs of `Type System` and `Type Subsystem` is associated with sub-process `V` (cf. Step 2a). Further, external state transitions are mapped to relation types.

In COREPRO, the OLCs for every object type and the external state transitions for every relation type form the *Life Cycle Coordination Model* (LCM) (cf. Fig. 2, Step 2a). Consequently, the LCM describes the dynamic aspects of the whole data model and constitutes the blueprint for creating the data-driven process structure.

On instance level, the *life cycle coordination structure* (LCS) describes the process structure for a particular data structure. While data model, data structure, and LCM are created manually, the LCS can be automatically generated based on these ingredients (cf. Step 2b in Fig. 2). The LCS includes an OLC for every object in the data structure. Likewise, for each relation in the data structure, external state transitions are inserted to the LCS. For example, for every `hasSubsystem` relation in the data structure from Fig. 2 (Step 1b), the associated external state transitions (with the associated sub-process `V`) are inserted. The result is an enactable process structure describing the dynamic aspects of the given data structure. Further details are presented in the following sections.

3 Modeling and Instantiation of Data Structures

In COREPRO, a domain specific data structure establishes the basis for creating data-driven process structures. COREPRO enables the definition of dynamic aspects of objects and relations at model level. Therefore, the definition of a data model may consist of object and relation types (cf. Fig. 3a). Based on this, data structures can then be created by instantiating specific objects and relations (cf. Fig. 3b).

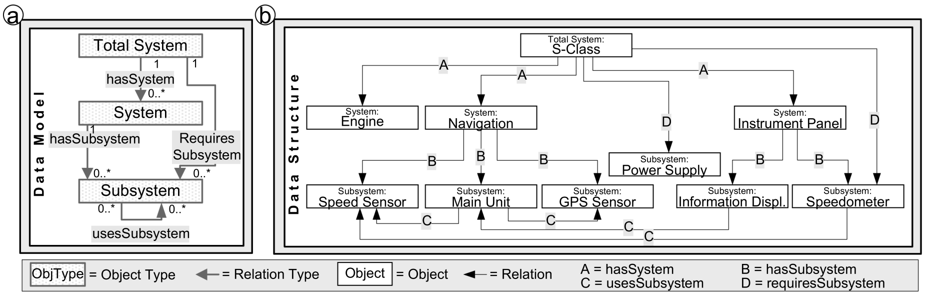


Fig. 3. a) Data Model and b) a Possible Instantiation (Data Structure)

3.1 Creation of a Data Model

Generally, a *data meta model* provides the constructs for describing a *data model* [12,13,14]. In COREPRO, we use a simple data meta model, comprising *object* and *relation types*[1]. An object type represents, for example, an abstract or physical product component that is part of the logical structure of a car. A relation type expresses a single relationship between two object types (including cardinality). A data model comprises object and relation types, and describes *how* objects are structured in a specific domain (cf. Fig. 3a). Based on the restrictions of the data model, several *data structures* (i.e., instances of the data model) can be created, such as product structures for different car series.

Generally, multiple relation types can be defined between two object types. Further, we allow defining recursive relation types, which can be used to realize relations between objects of the same object type on instance level.

Definition 1 (Data Model). *Let $\mathcal{T}$ be the set of all object types and let $\mathcal{R}$ be the set of all relation types. Then: A data model is a tuple $dm = (OT, RT)$ where*

– *$OT \subseteq \mathcal{T}$ comprises a set of object types*
– *$RT \subseteq OT \times \mathcal{R} \times OT$ comprises a set of binary relation types defined for object types*
– *$card : RT \mapsto \mathbb{N}_0 \times \mathbb{N}_0$ with $card(ot_1, rt, ot_2) = (min_{rt}, max_{rt})$ assigns to each relation type $rt \in RT$ a minimal and maximal cardinality.*

3.2 Creation of the Data Structure

A data structure contains *objects* and *relations*, which constitute instances of the object and relation types defined by the data model. In Fig. 3b, for example, the `Total System` type is instantiated once, while the `System` type is instantiated three times (`Engine`, `Navigation`, and `Instrument Panel`). The cardinalities associated with relation types restrict the number of concrete relations between objects. Accordingly, a `Subsystem` object can only be related to one `System`

[1] The data meta model neglects descriptive object attributes since they do not influent the generation of the requested process structure.

object via the `hasSubsystem` relation (cf. Fig. 3b). Note that the data structure from Fig. 3b is a simplified example. In Sect. 5 we indicate, that such a data structure may comprise a high number of instantiated objects in practice.

Definition 2 (Data Structure). *Let $dm = (OT, RT)$ be a data model. Then: A data structure created from dm is a tuple $ds = (O, R)$ with*

- *O is a set of objects where each object $o \in O$ is associated with an object type $objtype(o) \in OT$*
- *$R \subseteq O \times RT \times O$ is a set of object relations meeting the cardinality constraints defined by the data models. Each relation $r = (o_1, rt, o_2) \in R$ has a relation type $rt = reltype(r)$ with $(objtype(o_1), rt, objtype(o_2)) \in RT$.*

4 Integration of Data and Processes

After having defined how a data structure is modeled, we need to specify its relationship to the process structure. To allow for reuse, we describe this relationship at the model level; i.e., we define the dynamic aspects for the data model and translate them to the instance level afterwards. Therefore, an *object life cycle* (OLC) describes the dynamic aspects of an object type and an OLC dependency defines the dynamic aspects of a relation type. The *life cycle coordination model* (LCM) comprises the OLCs for every object type and the OLC dependencies for every relation type. Consequently, the LCM describes the dynamic aspects of the whole data model. On instance level, in turn, the LCM constitutes the basis for creating *life cycle coordination structures* (LCS) for given data structures. Altogether, the LCS defines the dynamic aspects of the underlying data structure and represents the data-driven process structure.

4.1 Modeling of the Dynamic Aspects of Single Object Types

Every object type is associated with an OLC, which constitutes a labeled transition system describing *object states* and *internal state transitions* (cf. Fig. 4). An internal state transition can be associated with a sub-process modifying the object (and thus inducing a state change). This sub-process becomes enacted when the source state of the transition is enabled. After having executed it, the source state of the transition is disabled and the target state becomes enabled. Hence, the definition of the OLC constitutes the mediator for associating stateful objects with modifying sub-processes[2].

To realize non-deterministic processing, the definition of conditional (i.e., non-deterministic) internal state transitions is possible in COREPRO. All internal state transitions with same OLC state as source are associated with the same sub-process, but can be bound to (different) sub-process results, i.e., exit codes (e.g., *finished with errors*). Depending on the concrete sub-process result, always one internal state transition is triggered at runtime. In Fig. 4, for example,

[2] Stateless objects are also supported. Their OLCs include one internal state transition (with an associated sub-process) from the start to the end state.

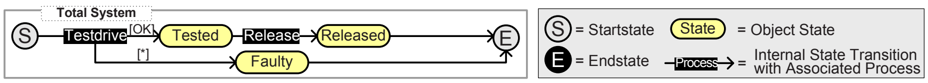


Fig. 4. Object Life Cycle with Conditional State Transitions

the result of the `Testdrive` sub-process determines, whether the state `Tested` or `Faulty` is enabled. The default transition to state `Faulty` (indicated by `*`) becomes activated in case of uncovered sub-process results. A conditional internal transition may also be used for modeling loops within an OLC.

Definition 3 (Object Life Cycle). *Let $dm = (OT, RT)$ be a data model and let $ot \in OT$ be an object type. Then: The object life cycle of ot is a tuple $olc = (P, V, TS)$ where*

- *P is a set of sub-processes that can be applied to instances of object type ot and V is a set of possible sub-process results ($\sigma : P \mapsto \mathcal{P}(V)$ with $\sigma(p) \subseteq V$ is the set of possible results defined for sub-process p)*
- *$TS = (S, T, s_{start}, s_{end})$ is a labeled transition system, where*
  - *S is the set of states that can be reached by objects of type ot*
  - *$T \subseteq S \times (P \times V) \times S$ is a set of <u>internal state transitions</u> with*
    - *$t = (s, (p, v), s') \in T, \Rightarrow v \in \sigma(p)$; i.e., a state transition t is triggered by the completion of a sub-process p with particular result v*
    - *$\forall t_i = (s_i, (p_i, v_i), s'_i) \in T, i = 1, 2$ with $t_1 \neq t_2$ and $s_1 = s_2, \Rightarrow p_1 = p_2 \wedge v_1 \neq v_2$; i.e., if there are several state transitions with same source state s, all of them will be associated with the same sub-process p. The concrete target state is determined based on the sub-process result. In case of non-deterministic state transitions, there is a default transition that will be chosen if the associated sub-process delivers a result not covered by the other transitions.*
  - *$s_{start} \in S$ is the initial state and $s_{end} \in S$ is the final state of the transition system; s_{start} is the only state without incoming transitions and s_{end} is the only state without outgoing transitions.*

Let $\mathcal{OLC}$ be the set of all object life cycles. For $olc \in \mathcal{OLC}$, $s_{start}(olc)$ denotes the start and $s_{end}(olc)$ the end state of the respective transition system.

4.2 Modeling of the Dynamic Aspects of the Data Model

Modeling the dynamic aspects of single object types is only one part of the challenge. To define the processing of the whole data model, we also have to specify the dynamic aspects of relation types. Relation Types are associated with *OLC dependencies* which synchronize the OLCs of related objects. An *OLC dependency* comprises several *external state transitions* between the states of the dependent OLCs. The resulting structure is denoted as *life cycle coordination model* (LCM). A LCM describes the dynamic aspects of the data model by integrating the OLCs associated with object types as well as the OLC dependencies

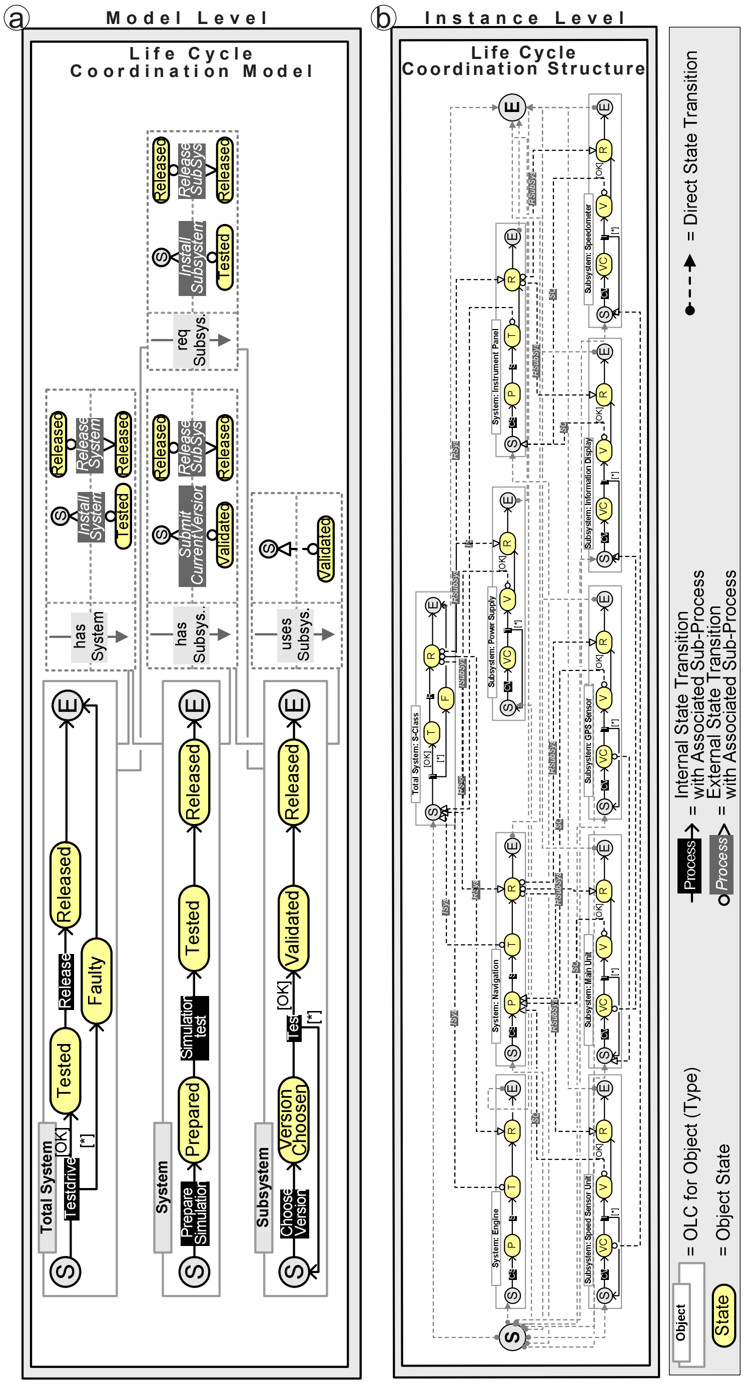


Fig. 5. Example for a) LCM and b) generated LCS

Table 1. Classification of State Transitions in COREPRO

Type	**Meaning**	**Operational Semantics**
`Internal State Transition`	Connects two states within one OLC	Fires after sub-process execution dependent on sub-process result
`External State Transition`	Connects two states from different OLCs	Fires after sub-process execution
`Direct State Transition`	Connects LCS start state with start state of an OLC (end states accordingly)	Fires immediately

associated with relation types (Fig. 5a presents the LCM for the data model from Fig. 3a).

Like internal state transitions (within an OLC), an external state transition can be associated with a sub-process. As an example, consider the OLC dependency of the relation type `hasSystem` in Fig. 5a. This dependency consists of two external state transitions, which synchronize (1) the `start` state of the `Total System` OLC with the `Tested` state of the `System` OLC and (2) the `Release` state of the `System` OLC with the `Release` state of the `Total System` OLC. The sub-process associated with the external state transition (e.g., sub-process `InstallComponent`) can be considered as *synchronizing (sub-)process*, which operates on both related object types.

Definition 4 (OLC Dependency). *Let* $olc_i = (P_i, V_i, TS_i), i = 1, 2$ *be two different object life cycles with* $TS_i = (S_i, T_i, s_{start}, s_{end})$ *(cf. Def. 3). Then: An OLC dependency between* olc_1 *and* olc_2 *is a tuple* $olc_{Dep} = (Id, P, EST)$ *where*

- Id *is the identifier of the dependency*
- P *is a set of sub-processes that can operate on both object types*
- EST *is a set of* <u>*external state transitions*</u> *with*
  $est = (s, p, s') \in EST \Leftrightarrow (s \in S_1 \wedge s' \in S_2) \vee (s' \in S_1 \wedge s \in S_2)$.

$\mathcal{OLC}_{\mathcal{DEP}}$ *denotes the set of all OLC dependencies. For an OLC dependency* $olc_{Dep} \in \mathcal{OLC}_{\mathcal{DEP}}$*, let* $est(olc_{Dep})$ *denote the set of related external state transitions.*

It is important to mention that internal and external state transitions differ in their operational semantics (cf. Table 1). During runtime, the concurrent processing of different objects is required to enhance process efficiency (e.g., by supporting concurrent engineering techniques). In COREPRO, this is realized by concurrently enacting different OLCs while avoiding concurrency within an OLC[3]. Both, internal and external state transition become activated (i.e., their sub-processes are started) when the source state of the transition is entered. While the completion of the sub-process of an internal state transition induces the deactivation of the source state and the activation of the target state, the completion of the sub-process of an external state transition does not imply any state change in the source OLC.

[3] Concurrently activated states are not allowed within one OLC since this has not been a requirement in our case studies.

Further, the target state is activated if and only if the sub-processes of (1) one incoming internal state transition and (2) all incoming external transitions are fired. This rule allows for concurrently activated states within different OLCs of an LCS, while it prevents concurrently activated states within a single OLC. Due to the lack of space, we omit a formal specification of the operational semantics of internal and external state transtions.

Definition 5 (Life Cycle Coordination Model). *Let $dm = (OT, RT)$ be a data model and let P be a set of sub-processes. Then: The life cycle coordination model associated with dm is a tuple $lcm = (olc, olc_{DEP})$ where*

– *$olc : OT \mapsto \mathcal{OLC}$ assigns to each object type $ot \in OT$ (of the data model) an object life cycle $olc(ot) \in \mathcal{OLC}$*
– *$olc_{DEP} : RT \mapsto \mathcal{OLC}_{DEP}$ assigns to each relation type $rt = (ot1, rt, ot2) \in RT$ an OLC dependency $olc_{DEP}(rt)$ for the object life cycles $olc(ot1)$ and $olc(ot2)$ of the object types $ot1$ and $ot2$.*

Regarding the execution of created process structures, it is important to guarantee soundness, i.e., to ensure that data-driven process structures terminate with a correct end state. Deadlocks might occur (1) when external state transitions are starting from non-deterministic states, and (2) when external state transitions are forming cycles. The first situation can be avoided during runtime, for example, using deadpath elimination techniques. The second situation can be recognized during buildtime by analyzing the process structure. Analyzing large data-driven process structures, however, generates high efforts. COREPRO enables checking soundness on model level and guarantees soundness for every data-driven process structure that bases on a sound LCM.

To check soundness of a LCM, all OLCs and OLC dependencies (i.e., their external state transitions) of the LCM are composed. In addition, a unique start state is added and connected with all start states of the OLCs via *direct state transitions* (cf. Fig. 6). Accordingly, all OLC end states are connected with a unique end state. Direct state transitions are concurrently triggered and lead to

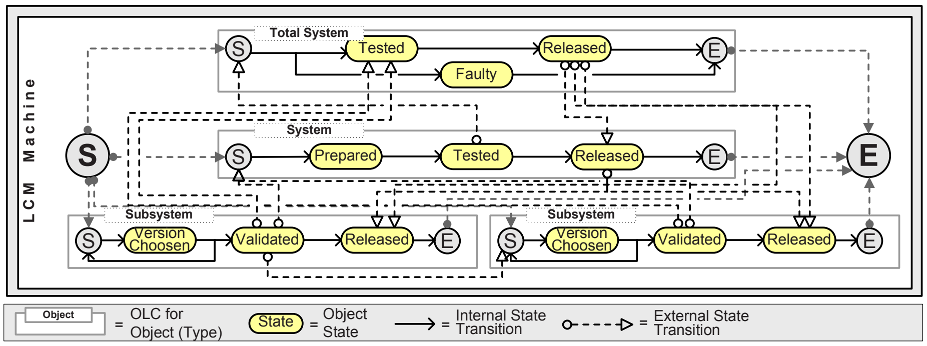


Fig. 6. LCM Machine for the LCM from Fig. 5a

the deactivation of the source state and the activation of the target state (cf. Table 1). We denote the extended transition system resulting from this as *LCM machine*. The LCM machine constitutes a LCS for a data structure where every element and relation type is instantiated once (to map OLC dependencies for recursive relation types, it is necessary to contemplate two OLCs for the object type associated with the recursive relation type). Thereby, efforts for soundness checks do not rise with the size of the instantiated process structure but only depend on the size of the LCM machine. As example consider Fig. 6, which shows the LCM machine for the LCM depicted in Fig. 5a. Since sub-processes connected with state transitions do not affect soundness checks (we presume sound sub-processes), they can be neglected when analyzing soundness.

Definition 6 (Soundness of the LCM). *A LCM machine is sound if each state of the LCM machine (including its end state) can be enabled by direct or internal state transitions beginning with the start state of the LCM machine and every external transition becomes activated (or deactivated) then.*

4.3 Creating the Life Cycle Coordination Structure

So far, we have introduced the data part which comprises the data model and the data structure, and the LCM (consisting of OLCs and OLC dependencies) which integrates the data model and the sub-processes. Based on this, the data-driven process structure, i.e., the *life cycle coordination structure* (LCS), can be automatically derived for respective data structures. The LCS comprises a start and an end state, an OLC instance for every object in the data structure, and external state transitions between these OLC instances according to the relations defined between the objects (cf. Fig. 5b).

Definition 7 (Life Cycle Coordination Structure). *Let $dm = (O, R)$ be a data structure and let $lcm = (olc, olc_{DEP})$ be a life cycle coordination model. Then: A life cycle coordination structure based on dm and lcm is a tuple $lcs = (olc_{inst}, est_{inst}, s_{start}, s_{end}, ST, ET)$ where*

- *$olc_{inst} : O \mapsto \mathcal{OLC}$ assigns to each object $o \in O$ an instance of the associated object life cycle $olc_{inst}(o) = olc(objtype(o))$*
- *$est_{inst} : R \mapsto \mathcal{OLC_{DEP}}$ assigns to each relation $r \in R$ the associated external state transitions $est_{inst}(r) = est(olc_{DEP}(reltype(r)))$*
- *s_{start} denotes the initial state and s_{end} the final state*
- *ST is the set of direct state transitions connecting the start state of the LCS with the start states of the instantiated object life cycles*
- *ET is the set of direct state transitions connecting the end states of the instantiated object life cycles with the end state of the LCS.*

The operations for creating a LCS are defined in Table 2. Based on a data structure and an LCM, three steps become necessary to generate the LCS. Algorithm 1 describes these steps in detail:

1. For every object in the data structure, the OLC associated with the corresponding object type is instantiated.

Table 2. Operations for Creating an LCS

Operation	**Effect**
`createLCS`	Creates a new LCS
`insertStartState(lcs)`	Inserts the initial state s_{start} to the given `lcs`
`insertEndState(lcs)`	Inserts the final state s_{end} to the given `lcs`
`insertOLC(lcs,olc)`	Inserts an instance of the given `OLC` to the given `lcs`
`insertExtTrans(lcs,s,p,s')`	Inserts an external state transition from state `s = (Transitionsystem, State)` to state `s' = (Transitionsystem, State)` with the associated sub-process `p` to the given `lcs`
`insertDirTrans(lcs,s,s')`	Inserts a direct state transition from state `s = (Transitionsystem, State)` to state `s' = (Transitionsystem, State)` to the given `lcs`

2. For every relation in the data structure, the OLC dependencies associated with the corresponding relation type (i.e., their external state transitions) are inserted to connect states of the dependent OLCs.
3. Direct state transitions are inserted, which connect the LCS start state with all OLC start states and all OLC end states with the LCS end state.

As result, we obtain the complete LCS representing the logical view on the data-driven process structure (cf. Fig. 5b). Such LCS can be transformed to activity-centered process representations, like BPMN or WS-BPEL.

Checking soundness of an LCS comprising hundreds up to thousands of sub-processes and (external) state transitions is a complex task to accomplish. COREPRO enables soundness checking on model level and ensures that every LCS created on basis of a sound LCM is sound as well (cf. Theorem 1).

Theorem 1 (Soundness of the LCS). *Assume that an LCS has been created with Alg. 1 with a particular data structure and an LCM as input. Then: If the LCM machine of the LCM is sound (cf. Definition 6), the created LCS is sound as well.*

Theorem 1 can be inductively proven: The LCM machine constitutes a sound LCS for a data structure where every object type and every relation type are

```
Input: $DS = (O, R)$, $LCM = (OLC, OLC_{DEP})$
Output: $lcs = (OLC_{inst}, EST_{inst}, s_{start}, s_{end}, ST, ET)$
// Initialize the LCS and insert start and end state
lcs := createLCS; s := insertStartState(lcs); e := insertEndState(lcs);
// Insert an OLC instance for every instantiated object and connect it with the start
    and end state of the LCS via directed state transitions
forall $obj \in O$ do
    olc := insertOLC(lcs,$olc(objtype(obj))$);
    insertDirTrans($lcs$, $(lcs, s_{start})$, $(olc, s_{start}(olc))$);
    insertDirTrans($lcs$, $(olc, s_{end}(olc))$, $(lcs, s_{end})$);
// Insert external state transitions for each instantiated relation
forall $rel = (o_1, rt, o_2) \in R$ do
    // Insert external state transitions between the OLC of $o_1$ and the OLC of $o_2$
    forall $est = (s_1, p, s_2) \in olc_{DEP}(rt)$ do
        insertExtTrans($lcs$, $(olc_{inst}(o_1), s_1)$, $p$, $(olc_{inst}(o_2), s_2)$);
return(lcs);
```

Algorithm 1. Generation of the Life Cycle Coordination Structure

instantiated once. When adding one additional object and all corresponding relations to other objects, we can prove that this leads to a sound LCS again ($n = 1$). Making this assumption for n additional objects, we can show that soundness can still be guaranteed when adding a further object and corresponding relations ($n \rightarrow n + 1$). Due to lack of space, we omit further details.

4.4 Change Scenarios

When dealing with data-driven process structures, change management becomes an important issue. Adaptations of process structures become necessary, for example, when the underlying data structure is changed (e.g., when adding a new object). In COREPRO, such changes can be specified at the data level and are then automatically translated into corresponding adaptations of the process structure. Compared to conventional approaches (e.g., the manual adaptation of activity-centered process structure representations), adaptation efforts can be significantly reduced. To illustrate this, we sketch three change scenarios in which users (e.g., engineers) adapt the (product) data structure.

Removing an object
Example: The `Speed Sensor` subsystem shall not be processed any longer, i.e., the `Speed Sensor` and all its relations to or from other objects are removed from the data structure (cf. Fig. 3b).
Conventional approach: Manually removing associated sub-processes and their incoming and outgoing synchronization links from the process structure.
COREPRO procedure: The corresponding OLC and the external state transitions are automatically removed from the LCS.

Removing a relation
Example: The `Main Unit` does not use the `Speed Sensor` any longer, i.e., the relation `usesSubsystem` between the `Main Unit` and the `Speed Sensor` object is removed from the data structure (cf. Fig. 3b).
Conventional approach: Manually removing the sub-process dependencies which are no longer necessary from the process structure (i.e., synchronization links of the sub-processes modifying the `Main Unit` and the `Speed Sensor` subsystem).
COREPRO procedure: The corresponding external state transitions are automatically removed from the LCS.

Adding an object
Example: A new `Head-Up Display` subsystem shall be processed as part of the `Navigation` system, i.e., a new object is added to the data structure and related to the `Navigation` object (cf. Fig. 3b).
Conventional approach: Manually inserting the corresponding sub-processes and associated synchronization links.
COREPRO procedure: The corresponding OLC and the external state transitions are automatically added to the LCS.

Taking these scenarios, COREPRO allows users to apply process structure changes at a high level of abstraction. Using conventional, activity-driven approaches for process modeling, process structures would have to be manually adapted in case of a data structure change. That requires extensive process knowledge and necessitates additional soundness checks. By contrast, COREPRO significantly reduces efforts for adaptation. The process structure can be adapted without comprehensive process knowledge by simply changing the data structure. The soundness of the resulting process structure is assured, since the model level (i.e., the data model and the LCM) remains unchanged.

5 Practical Impact

To indicate the practical benefit of our modeling approach, we introduce a model calculation for a characteristic process from the car development domain: the release management (RLM) process for electrical systems [1,3]. The calculation (cf. Table 3) bases on the experiences we gained from case studies in this domain.

The release of an electrical system encompasses 200 to 300 components (depending on the car series), which are divided in root components and their variants (e.g., driver's airbag as root component and passenger's airbag as its variant). Root components are further divided into categories requiring different processing. For example, releasing a multimedia component requires different sub-processes when compared to a security related component. Additionally, components are grouped into systems (e.g., navigation system covers several components) to integrate logically coherent components. Finally, systems are collected in total systems, which represent the car series to be developed (e.g., S-Class). Altogether, this leads to the definition of a data model with about 20 object types and 25 relation types connecting them.

On instance level, the relation types with a `1:n` cardinality lead to more than 200 instantiated relations. Additionally, there exist dependencies between components that exchange signals and messages. These relation types are defined with an `n:m` cardinality leading to more than 400 relations. OLCs for the different object types coordinate 5 (for components) to 20 (for systems) sub-processes.

In constrast to conventional modeling, the creation of the process structure constitutes an automated task. In total, the LCS contains 200 to 300 OLCs (according to the number of objects) with more than 1300 sub-processes. Relation types encompass 2 to 6 external state transitions (cf. Table 3) leading to more than 1500 external state transitions within the generated LCS.

Table 3. Projection of the Modeling Efforts Reduction

Model			
Data Model		**LCM**	
Object Types	Relation Types	Sub-Processes	Ext. Transitions per Relation Type
20	25	5-20	2-6
Instance			
Data Structure		**LCS**	
Objects	Relations	Sub-Processes	Ext. Transitions
200-300	>600	>1300	>1500
Modeling Efforts Reduced by			
>90%	>95%	>98%	>99%

The potential for reducing modeling efforts when using the instantiation mechanism of COREPRO depends on the ratio of object types to objects. Even though the calculation bases on a moderate estimate, it indicates that the modeling efforts for RLM processes can be significantly reduced by more than 90%. The benefit even increases considering the fact that soundness checks (cf. Sect. 4.2) and changes (cf. Sect. 4.4) become less complex.

6 Related Work

Activity-driven approaches for process modeling do not focus on the automated creation of large process structures comprising multiple sub-processes. Interaction graphs [15] and choreography definition languages (e.g., [16]), for example, are activity-centered approaches to specify the choreography between distributed sub-processes. The data-driven derivation of sub-process dependencies is not covered. Other approaches partially enable the data-driven creation of process structures. For example, multiple instantiation of activities based on simple data structures (set, list) [17,18] is supported by UML 2.0 activity diagrams (*Expansion Region*) [14] and BPMN (*Multiple Instances*) [19]. They enable iterated or concurrent execution of the same activity for each element given by a flat data container. Utilization of respective data structures raises further options, such as data-driven process control with exception handling mechanisms [20]. However, these approaches aim at the sequential or concurrent execution of multiply instantiated activities. COREPRO, by contrast, focuses on the definition of arbitrary complex data structures and their association with (sub-)processes. The data-driven process structure in Fig. 7, for example, realizes the interleaved synchronization of sub-processes. It represents a list structure where sub-processes are executed for every list element. In this context, COREPRO also allows for the realization of anticipation concepts [21]: Regarding the generated LCS, `Process A` for `Element 2` can be started even though the processing of `Element 1` has not been finished.

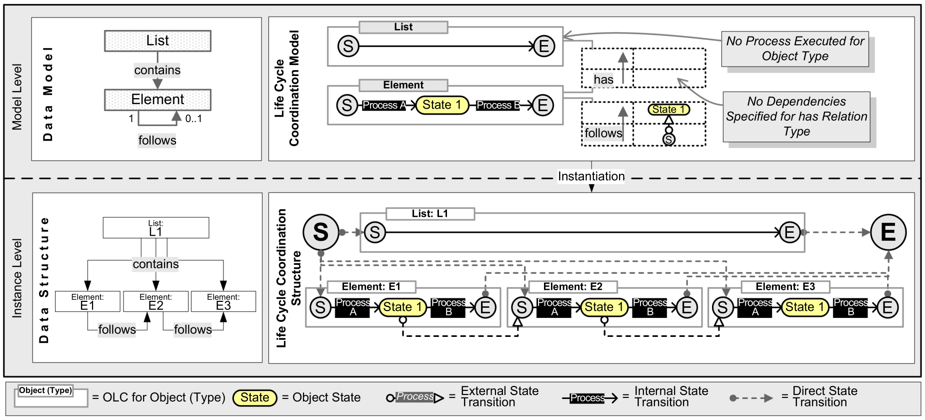


Fig. 7. Interleaved Processing of a List

Approaches for explicitly generating process structures based on bills of material are described in [5,7]. The idea of coordinating activities based on data dependencies also constitutes the basis of the *Case Handling* paradigm [6]. The idea is to model a process structure by relating activities to the data flow. The concrete activity execution order at runtime then depends on the availability of data. Another approach integrating control and data flow is provided by *AHEAD* [22], which offers dynamic support for (software) development process structures. The approach enables the integration of control and data flow, by relating activities to the objects defined in the data model. Based on this information, dynamic task nets are generated. The goal of respective data-driven approaches is the precise mapping of object relations to sub-process dependencies. Though the object relations indicate sub-process dependencies (cf. Sect. 2), the information given by relations is insufficient for their direct mapping to synchronization edges for three reasons. First, *several* sub-processes may modify one object, whereas the relation itself does not reflect which of these sub-processes have to be synchronized. Second, the relations do not provide sufficient information about the direction of synchronization dependencies. In Fig. 1, for example, the relation points from the `Speed Sensor` to the `Navigation` object, while the synchronization dependencies between their sub-processes point in both directions. Third, it is also requested to associate synchronization dependencies with the enactment of synchronizing sub-processes (cf. Sect. 4.2).

A general approach following the idea of modeling life cycles and relating them is *Object/Behavior Diagrams*. The concept allows for the object-oriented definition of data models which can be enhanced by runtime aspects [8]. The behavior is defined for every object within a Petri Net relied life cycle diagram. Another approach using life cycles for describing operational semantics of business artifacts (semantic objects) is *Operational Specification* (OpS) [11]. The collection of all objects and their life cycles specifies the operational model for the entire business. The *Object-Process Methodology* (OPM) is an object-oriented approach from the engineering domain. It focuses on connecting objects (or object states) and processes by procedural links [10]. *Team Automata* provide a formal method to describe the connection of labeled transition systems (automata) via external actions associated with (internal) transitions [9]. Automata including transitions with the same external action perform them simultaneously. The idea is adopted in [23] where Team-Automata are structured in an object-oriented way. Its synchronization mechanisms are based on events. These approaches rather focus on an activity-driven specification of dependencies (based on events) than on the consideration of data relations for process structure generation. In contrast to event-based synchronization, external state transitions can be added, removed or disabled (e.g., in order to avoid deadlocks) without changing the dynamic aspects of the object itself (i.e., the OLC).

7 Summary and Outlook

The COREPRO approach offers promising perspectives with respect to the modeling and coordination of large data-driven process structures consisting of numerous sub-processes and their interdependencies. COREPRO supports the loose coupling of data and sub-processes by defining life cycles of data objects, and it provides a well-defined mapping of object relations to OLC dependencies. Further, COREPRO distinguishes between model and instance level, which enables a high level of abstraction, extensibility and reuse. In particular, modeling and change effectiveness are significantly enhanced by

– introducing model-driven design in conjunction with an instantiation mechanism for data-driven process structures
– enabling the instantiation of different data structures and automatically generating respective data-driven process structures
– integrating data and processes which allows users without process knowledge to adapt the process structures by changing the data structure.

Another important issue to be considered is the need for flexibility at runtime, such as applying structural changes during enactment (cf. Sect. 4.4). This becomes necessary, for example, when the number of objects or relations between them is not (exactly) known at buildtime. Due to the many sub-process dependencies, uncontrolled runtime changes may lead to inconsistencies not only within single OLCs, but also within the whole LCS. In addition to structural changes, we also have to consider state changes. To realize iterative development processes, for example, data structures need to be processed several times. However, that necessitates the (partial) utilization of previous processing states of objects; i.e., object states which were already activated before execution, have to be retained. For example, product components which have already been tested and which remain unchanged do not need to be tested again. Applying such changes and supporting exceptional situations (e.g., abnormal termination of a sub-process or backward jumps within an OLC) while preserving consistency is a challenging problem [2]. Solutions for runtime scenarios and exception handling are also addressed by COREPRO and will be presented in future publications.

We have implemented major parts of the presented modeling concepts in a prototype, which we use for a first proof-of-concept case study in car development [24]. The approach will be applied for modeling, coordinating and maintaining data-driven process structures in the automotive industry. However, the presented concept is not specific to the engineering domain. We also plan to evaluate COREPRO in the healthcare domain, where the approach shall be used to model medical treatment processes [25].

References

1. Müller, D., Herbst, J., Hammori, M., Reichert, M.: IT support for release management processes in the automotive industry. In: Washio, T., Sakurai, A., Nakajima, K., Takeda, H., Tojo, S., Yokoo, M. (eds.) New Frontiers in Artificial Intelligence. LNCS (LNAI), vol. 4102, pp. 368–377. Springer, Heidelberg (2006)
2. Müller, D., Reichert, M., Herbst, J.: Enabling flexibility of data-driven process structures. In: Huang, D.-S., Li, K., Irwin, G.W. (eds.) ICIC 2006. LNCS, vol. 4103, pp. 181–192. Springer, Heidelberg (2006)

3. Bestfleisch, U., Herbst, J., Reichert, M.: Requirememts for the workflow-based support of release management processes in the automotive sector. In: ECEC, pp. 130–134 (2005)
4. VDI: VDI Systematic Approach to the Design of Technical Systems and Products. Beuth Verlag (1987) (VDI Guidelines (2221)
5. Aalst, W.: On the automatic generation of workflow processes based on product structures. Comput. Ind. 39(2), 97–111 (1999)
6. Aalst, W., Berens, P.J.S.: Beyond workflow management: Product-driven case handling. In: GROUP, pp. 42–51 (2001)
7. Reijers, H., Limam, S., Aalst, W.: Product-based workflow design. MIS 20(1), 229–262 (2003)
8. Kappel, G., Schrefl, M.: Object/behavior diagrams. In: ICDE, pp. 530–539 (1991)
9. Ellis, C.A.: Team automata for groupware systems. In: Group, pp. 415–424. ACM, New York (1997)
10. Dori, D.: Object-process methodology as a business-process modelling tool. In: ECIS (2000)
11. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
12. Chen, P.: The entity-relationship model - toward a unified view of data. ACM Transactions on Database Systems 1(1), 9–36 (1976)
13. Jackson, M.A.: Principles of Program Design. Academic Press, London (1975)
14. OMG: UML Superstructure proposal 2.0 (2003)
15. Heinlein, C.: Workflow and process synchronization with interaction expressions and graphs. In: ICDE, pp. 243–252 (2001)
16. W3C: WS-CDL 1.0 (2005)
17. Aalst, W., Hofstede, A., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
18. Guabtni, A., Charoy, F.: Multiple instantiation in a dynamic workflow environment. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 175–188. Springer, Heidelberg (2004)
19. BPMI: Business process modeling notation specification (BPMN) (2006)
20. Rinderle, S., Reichert, M.: Data-driven process control and exception handling in process management systems. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 273–287. Springer, Heidelberg (2006)
21. Grigori, D., Charoy, F., Godart, C.: Coo-flow: A process technology to support cooperative processes. IJSEKE 14(1), 61–78 (2004)
22. Jäger, D., Schleicher, A., Westfechtel, B.: AHEAD: A graph-based system for modeling and managing development processes. In: Münch, M., Nagl, M. (eds.) AGTIVE 1999. LNCS, vol. 1779, pp. 325–339. Springer, Heidelberg (2000)
23. Engels, G., Groenewegen, L.: Towards team-automata-driven object-oriented collaborative work. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.) Formal and Natural Computing. LNCS, vol. 2300, pp. 257–276. Springer, Heidelberg (2002)
24. Müller, D., Reichert, M., Herbst, J., Poppa, F.: Data-driven design of engineering processes with COREPRO$_{Modeler}$. In: WETICE (ProGility) (2007)
25. Lenz, R., Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives. Business Process Management 61, 39–58 (2007)

Supporting Ad-Hoc Changes in Distributed Workflow Management Systems

Manfred Reichert[1] and Thomas Bauer[2]

[1]Informaton Systems Group, University of Twente, The Netherlands
m.u.reichert@cs.utwente.nl
[2]Dept. GR/EPD, DaimlerChrysler AG Group Research, Germany
thomas.tb.bauer@daimlerchrysler.com

Abstract. Flexible support of distributed business processes is a characteristic challenge for any workflow management system (WfMS). Scalability at the presence of high loads as well as the capability to dynamically adapt running process instances are essential requirements. Should the latter one be not met, the WfMS will not have the necessary flexibility to cover the wide range of process-oriented applications deployed in many organizations. Scalability and flexibility have, for the most part, been treated separately in literature thus far. Even though they are basic needs for a WfMS, the requirements related with them are totally different. To achieve satisfactory scalability, on the one hand the system needs to be designed such that a workflow (WF) instance can be controlled by several WF servers that are as independent from each other as possible. Yet dynamic WF changes, on the other hand, necessitate a (logical) central control instance which knows the current and global state of a WF instance. This paper presents methods which allow ad-hoc modifications (e.g., to insert, delete, or shift steps) to be correctly performed in a distributed WfMS; i.e., in a WfMS with partitioned WF execution graphs and distributed WF control. It is especially noteworthy that the system succeeds in realizing the full functionality as given in the central case while, at the same time, achieving favorable behavior with respect to communication costs.

1 Introduction

Workflow management systems (WfMS) enable the definition, execution, and monitoring of computerized business processes. Very often, a centralized WfMS shows deficits when it is confronted with high loads or when the business processes to be supported span multiple organizations. As in several other approaches (e.g. [9,15]), in the ADEPT project, we have met this particular demand by realizing a distributed WfMS made up of several workflow (WF) servers. WF schemes may be divided into several partitions such that related WF instance may be controlled "piecewise" by different WF servers in order to obtain a favorable communication behavior [3,5]. Such a distributed WF execution is also needed, for example, for the WF-based support of ubiquitous applications and their integration with backend systems.

Comparable to centralized WfMS (e.g., Staffware), a distributed WfMS must meet high flexibility requirements in order to cover the broad spectrum of processes we can find in large organizations [16,20,14]. In particular, at the WF instance level it must be possible to deviate from the pre-defined WF schema during runtime if required (e.g., by adding, deleting or moving process activities in the flow of control). As reported in literature (e.g., [14,19]), such ad-hoc WF changes become necessary to deal with exceptional or changing situations. Within the ADEPT project we have developed an advanced technology for the support of adaptive workflows. In paticular, ADEPT allows users (or agents) to dynamically modify a running WF instance without causing any run-time error or inconsistency in the sequel (e.g., deadlocks or program crashes due to activity invocations with missing input parameter data) [16,17].

In our previous work we considered distributed execution of partitioned WF schemes and ad-hoc modifications as separate issues. In fact, we have not systematically investigated how these two vital aspects of a WfMS interact. Typically such an investigation is not trivial as the requirements related to each of these two aspects are different: Ad-hoc WF instance modifications and the correct processing of the WF instance afterwards prescribe a logically central control instance to ensure correctness and consistency [16]. The existence of such a central instance, however, contradicts to the accomplishments achieved by distributed WF execution. The reason for this is that a central component decreases the availability of the WfMS and increases communication efforts between WF clients and the WF server. One reason for this lies in the fact that the central control instance must be informed of each and every change in the state of any WF instance. This state of the instance is needed to decide whether an intended modification is executable at all [16].

The objective of this paper is to introduce an approach for enabling ad-hoc modifications of single WF instances in a distributed WfMS; i.e., a WfMS with WF schema partitioning and distributed WF control. As a necessary prerequisite, distributed WF control must not affect the applicability of ad-hoc modifications; i.e., each modification, allowed in the central case, must be applicable in case of distributed WF execution as well. And the support of such ad-hoc modifications, in turn, must not impact distributed WF control. In particular, normal WF execution should not necessitate a great deal of additional communication effort due to the application of WF instance modifications. Finally, in the system to be developed, ad-hoc modifications should be correctly performed and as efficiently as possible. To deal with these requirements, it is essential to examine which servers of the WfMS must be involved in the synchronization of an ad-hoc modification. Most likely we will have to consider those servers currently involved in the control of the respective WF instance. These active servers require the resulting *execution schema* of the WF instance (i.e., the schema and state resulting from the ad-hoc modification) in order to correctly control it after the modification. Thus we first need an efficient approach to determine the set of active servers for a given WF instance. This must be possible without a substantial expense of communication efforts. In addition, we must clarify how the new execution schema of the WF instance, generated as a result of the ad-hoc modification, may be transmitted to relevant servers. An essential requirement is, thereby, that the amount of communication may not exceed acceptable limits.

Section 2 gives background information about distributed WfMS, which which is needed for the understanding of this paper. Section 3 describes how ad-hoc modifications are performed in a distributed WfMS, while Section 4 sets out how modified WF instances can be efficiently controlled in such a system. We discuss related work in Section 5 and end with a summary in Section 6.

2 Distributed Workflow Execution in ADEPT

Usually, WfMS with one central WF server are unsuitable if the WF participants (i.e., the actors of the WF activities) are distributed across multiple enterprises or organizational units. In such a case, the use of one central WF server would restrict the autonomy of the involved partners and might be disadvantageous with respect to respones times. Particularly, if the organizations are widespread, response times will significantly increase due to the long distance communication between WF clients and the WF server. In addition, owing to the large number of users and co-active WF instances typical for enterprise-wide applications, the WfMS is generally subjected to an extremely heavy load. This may lead to certain components of the system becoming overloaded. For all these and other reasons, in the distributed variant of ADEPT, a WF instance may not be controlled by only one WF server. Instead, its related WF schema may be partitioned at buildtime (if favorable), and the resulting partitions be controlled "piecewise" by multiple WF servers during runtime [1] [3] (cf. Fig. 1). As soon as the end of a partition is reached at run-time, control over the respective WF instance is handed over to the next WF server (in the following we call this *migration*).

When performing such a migration, a description of the state of the WF instance has to be transmitted to the target server before this WF server can take over control. This includes, for example, information about the state of WF activities as well as values for WF relevant data; i.e., data elements connected with output parameters of activities. (To simplify matters, in this paper we assume that WF templates (i.e., respective WF schemes) have been replicated and stored on all (relevant) WF servers of the distributed WfMS.)

To avoid unnecessary communication between WF servers, ADEPT allows to control parallel branches of a WF instance independently from each other – at least as no synchronization due to other reasons, e.g. a dynamic WF modification, becomes necessary. In the example given in Figure 1b, WF server s_3, which currently controls activity d, normally does not know how far execution has progressed in the upper branch (activities b and c). This has the advantage that the WF servers responsible for controlling the activities of parallel branches do not need to be synchronized.

[1] To achieve a better scalability we allow the same partition of different WF instances to be controlled by multiple WF servers (for details see [6]).

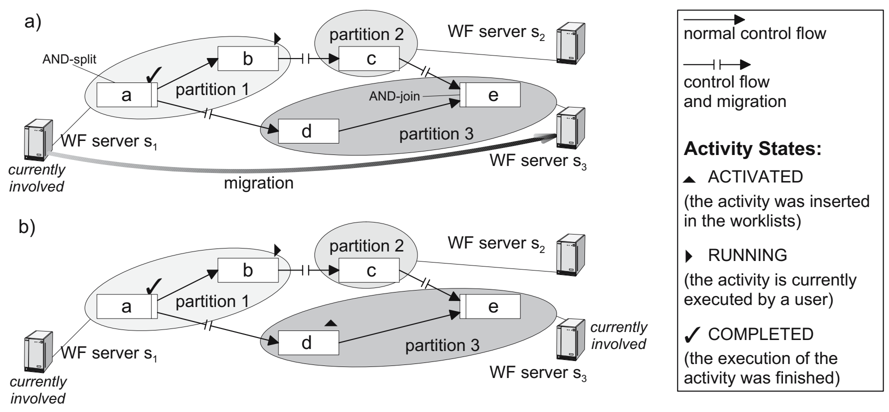


Fig. 1. a) Migration of a WF instance (from s_1 to s_3) and b) the resulting state of the WF instance

The partitioning of WF schemes and distributed WF control have been successfully utilized in other approaches as well (e.g. [9,15]).In ADEPT, we have targeted an additional goal, namely the minimization of communication costs. Concrete experiences we gained in working with commercial WfMS have shown that there is a great deal of communication between the WF server and its WF clients, oftentimes necessitating the exchange of large amounts of data. This may lead to the communication system becoming overloaded. Hence, the WF servers responsible for controlling activities in ADEPT are defined in such a way that communication in the overall system is reduced: Typically, the WF server for the control of a specific activity is selected in a way such that it is located in the subnet to which most of the potential actors belong (i.e., the users whose role would allow them to handle the activity). This way of selecting the server contributes to avoid cross-subnet communication between the WF server and its clients. Further benefits are improved response times and increased availability. This is achieved due to the fact that neither a gateway nor a WAN (Wide Area Network) is interposed when executing activities. The efficiency of the described approach – with respect to WF server load and communication costs – has been proven by means of comprehensive simulations and is outside the scope of this paper (see [4]).

Usually, servers are assigned to the activities of a WF schema already at build-time. However, in some cases this approach does not suffice to achieve the desired results. This may be the case, for example, if *dependent actor assignments* become necessary. Such assignments indicate, for example, that an activity n has to be performed by the same actor as a preceding activity m. Consequently, the set of potential actors of activity n is dependent on the concrete actor assigned to activity m. Since this set of prospective actors can only be determined at run-time, it would be beneficial to wait with WF server assignment until run-time as well. Then, a server in a suitable subnet can be selected; i.e., one that is most favorable for the actors defined. For this purpose, ADEPT supports so-called *variable server assignments* [5]. Here, server assignment expressions like "server in subnet of the actor performing activity m" are assigned to activities and then evaluated at run-time. This allows the WF server, which shall control the related activity instance, to be determined dynamically.

3 Ad-Hoc Modifications in a Distributed WfMS

In principle, in a distributed WfMS ad-hoc modifications of single WF instances have to be performed just as in a central system (for an example see Fig. 2). The WfMS has to check whether or not the desired modification is allowed on basis of the current structure and state of the concerned WF instance. If the modification is permissible (e.g., if the instance has not progressed too far in its execution), the related change operations will have to be determined and the WF schema belonging to the WF instance will be modified accordingly (incl. adaptations of the WF instance state if required).

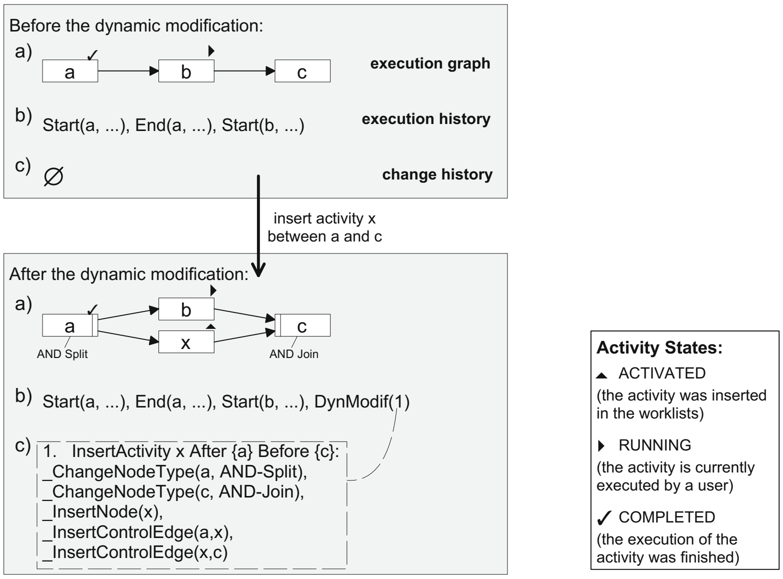

Before the dynamic modification:
a)
a
b
c
execution graph
b) Start(a, ...), End(a, ...), Start(b, ...)
execution history
c) ∅
change history
insert activity x between a and c
After the dynamic modification:
a)
a
b
x
c
AND Split
AND Join
b) Start(a, ...), End(a, ...), Start(b, ...), DynModif(1)
c) 1. InsertActivity x After {a} Before {c}:
_ChangeNodeType(a, AND-Split),
_ChangeNodeType(c, AND-Join),
_InsertNode(x),
_InsertControlEdge(a,x),
_InsertControlEdge(x,c)
Activity States:
ACTIVATED
(the activity was inserted in the worklists)
RUNNING
(the activity is currently executed by a user)
COMPLETED
(the execution of the activity was finished)

Fig. 2. (Simplified) example of an ad-hoc modification in a centralized WfMS with a) WF execution schema, b) execution history, and c) modification history

To investigate whether an ad-hoc modification is permissible in a distributed WfMS, first, the system needs to know the current global state of the (distributed) WF instance (or at least relevant parts of it). In case of parallel execution branches this state information may be distributed over several WF servers and therefore may have to be retrieved from these WF servers when a change becomes necessary.

This section describes a method for determining the set of WF servers on which the state information relevant for the applicability of a modification is located. In contrast to a central WfMS, in distributed WfMS it is generally not sufficient to modify the execution schema of the WF instance solely on the WF server responsible for controlling the modification. Otherwise, errors or inconsistencies may occur in the following, since other WF servers would use ”out-of-date” schema and state information when controlling the WF instance. Therefore, in the following, we show which WF servers have to be involved in the modification procedure and how corresponding protocols look like.

3.1 Synchronizing Workflow Servers During Ad-Hoc Modifications

An authorized user may invoke an ad-hoc modification on any WF server which (currently) controls the WF instance in question. Yet as a rule, this WF server alone may not always be able to correctly perform the modification. If other WF servers currently control parallel branches of the corresponding WF instance, state information from these WF servers may be needed as well. In addition, the WF server initiating the change process must also ensure that the corresponding modifications are taken over into the execution schemes of the respective WF instance, which are being managed by these other WF servers. Note that this becomes necessary to enable them to correctly proceed with the control flow in the sequel (see below). A naive solution would be to involve all WF servers of the WfMS by a broadcast. However, this approach is impractical in most cases as it is excessively expensive. In addition, all server machines of the WfMS must be available before an ad-hoc modification can be performed. Thus we have come up with three alternative approaches, which we explain and discuss below.

Approach 1: Synchronize all Servers Concerned With the WF Instance

This approach considers those WF servers which either have been or are currently active in controlling activities of the WF instance or which will be involved in the execution of future activities. Although the effort involved in communication is greatly reduced as compared to the naive solution mentioned above, it may still be unduly large. For example, communication with those WF servers which were involved in controlling the WF instance in the past and which will not participate again in future is superfluous. They do not need to be synchronized any more and the state information managed by them has already been migrated.

Approach 2: Synchronize Current and Future Servers of the WF Instance.

To be able to control a WF instance, a WF server needs to know its current WF execution schema. This, in turn, requires knowledge of all ad-hoc modifications performed so far. For this reason, a modification is relevant for those WF servers which either are currently active in controlling the WF instance or will be involved in controlling it in the future. Thus it seems to make sense to synchronize exactly these WF servers in the modification procedure. However, with this approach, problems may arise in connection with conditional branches.

For XOR-splits, which will be performed in the future, it cannot always be determined in advance which execution branch will be chosen. As different execution branches may be controlled by different WF servers, the set of relevant WF servers cannot be calculated immediately. Generally, it is only possible to calculate the set of the WF servers that will be potentially involved in this WF instance in the future. The situation becomes even worse if variable server assignments (cf. Sect. 2) are used. Then, generally, for a given WF instance it is not possible to determine the WF servers that will be potentially involved in the execution of future activities. The reason for this is that the run-time data of the WF instance, which is required to evaluate the WF server assignment expressions, may not even exist at this point in time. For example, in Figure 3, during execution of activity g, the WF server of activity j cannot be determined since the actor responsible for activity i has not been fixed yet. Thus the system will not always be able to synchronize future servers of the WF instance when an ad-hoc modification takes place. As these WF servers do not need to be informed about the modification at this time (since they do not yet control the WF instance), we suggest another approach.

Approach 3: Synchronize all Current Servers of the WF Instance

The only workable solution is to synchronize exclusively those WF servers currently involved in controlling the WF instance, i.e. the active WF servers. Generally, it is not trivial at all to determine which WF servers these in fact are. The reason is that in case of distributed WF control, for an active WF server of a WF instance the execution state of the activities being executed in parallel (by other WF servers) is not known. As depicted in Figure 3, for example, WF server s_4, which controls activity g, does not know whether migration $M_{c,d}$ has already been executed and, as a result, whether the parallel branch is being controlled by WF server s_2 or by WF server s_3. In addition, it is not possible to determine which WF server controls a parallel branch, without further effort, if variable server assignments are used. In Figure 3, for example, the WF server assignment of activity e refers to the actor of activity c, which is not known by WF server s_4. – In the following, we restrict our considerations to Approach 3.

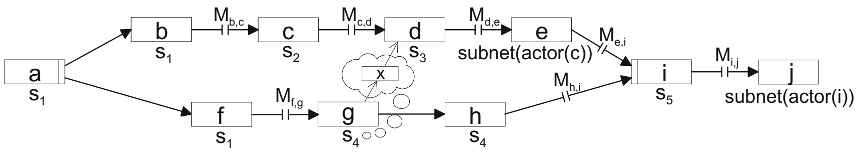


Fig. 3. Insertion of activity x between the activities g and d by the server s_4

3.2 Determining the Set of Active Servers of a Workflow Instance

As explained above, generally, a WF server is not always able to determine from its local state information which other WF servers are currently executing activities of a specific WF instance. And it is not a good idea to use a broadcast call to search for these WF servers, as this would result in exactly the same drawbacks as described for the naive solution at the beginning of Section 3.1. We, therefore, require an approach for explicitly managing the active WF servers of a WF instance. The administration of these WF servers, however, should not be carried out by a fixed (and therefore central) WF server since this might lead to bottlenecks, thus negatively impacting the availability of the whole WfMS.

For this reason, in ADEPT, the set of active WF servers (*ActiveServers*) is managed by a *ServerManager* specific to the WF instance. For this purpose, for example, the start server of the WF instance can be used as the *ServerManager*. Normally, this WF server varies for each of the WF instances (even if they are of the same WF type), thus avoiding bottlenecks.

The start WF server can be easily determined from the (local) execution history by any WF server involved in the control of the WF instance. The following subsections show how the set of active WF servers of a specific WF instance is managed by the *ServerManager*, how this set is determined, and how ad-hoc modifications can be efficiently synchronized.

Managing Active WF Servers of a WF Instance. As mentioned above, for the ad-hoc modification of a WF instance we require the set *ActiveServers*, which comprises all WF servers currently involved in the control of the WF instance. This set, which may be changed due to migrations, is explicitly managed by the *ServerManager*. Thereby, the following two rules have to be considered:

1. Multiple migrations of the same WF instance must not overlap arbitrarily, since this would lead to inconsistencies when changing the set of active WF servers.
2. For a given WF instance, the set *ActiveServers* must not change due to migrations during the execution of an ad-hoc modification. Otherwise, wrong WF servers would be involved in the ad-hoc modification or necessary WF servers would be left out.

As we will see in the following, we prevent these two cases by the use of several locks.[2] We now describe the algorithms necessary to satisfy these requirements. Algorithm 1 shows the way migrations are performed in ADEPT. It interacts with Algorithm 2 by calling the procedure *UpdateActiveServers* (remotely), which is defined by this algorithm. This procedure manages the set of active WF servers currently involved in the WF instance; i.e., it updates this set consistently in case of WF server changes.

[2] A secure behavior of the distributed WfMS could also be achieved by performing each ad-hoc modification and each migration (incl. the adaptation of the set *ActiveServers*) within a distributed transaction (with 2-phase-commit). But this approach would be very restrictive since during the execution of such an operation, "normal WF execution" would be prevented. That means, while performing a migration, the whole WF instance would be locked and, therefore, even the execution of activities actually not concerned would not be possible. Such a restrictive approach is not acceptable for any WfMS. However, it is not required in our approach and we realize a higher degree of parallel execution while achieving the same security.

Algorithm 1 illustrates how a migration is carried out. It is initiated and executed by a source WF server that hands over control to a target WF server. First, the *SourceServer* requests a non-exclusive lock from the *ServerManager*, which prevents the migration from being performed during an ad-hoc modification (cf. Algorithm 3). Then an exclusive, short-term lock is requested. This lock ensures that the *ActiveServers* set of a given WF instance is not changed simultaneously by several migrations within parallel branches. (Both lock requests may be incorporated into a single call to save a communication cycle.)

The *SourceServer* reports the change of the *ActiveServers* set to the *ServerManager*, specifying whether it remains active for the concerned WF instance (*Stay*), or whether it will not be involved any longer (*LogOff*). If, for example, in Figure 3 the migration $M_{b,c}$ is executed before $M_{f,g}$, the option *Stay* will be used for the migration $M_{b,c}$ since WF server s_1 remains active for this WF instance. Thus, the option *LogOff* is used for the subsequent migration $M_{f,g}$ as it ends the last branch controlled by s_1. The (exclusive) short-term lock prevents that these two migrations may be executed simultaneously. This ensures that it is always clear whether or not a WF server remains active for a WF instance when a migration has ended. Next, the WF instance data (e.g., the current state of the WF instance) is transmitted to the target WF server of the migration. Since this is done after the exclusive short-term lock has been released (by *UpdateActiveServers*), several migrations of the same WF instance may be executed simultaneously. The algorithm ends with the release of the non-exclusive lock.

Algorithm 1 (Performing a Migration)

```
input
    Inst: ID of the WF instance to be migrated
    SourceServer: source server of the migration (it performs this algorithm)
    TargetServer: target server of the migration
begin
    // calculate the ServerManager for this WF instance by the use of its execution history
    ServerManager = StartServer(Inst);
    // request a non-exclusive lock and an exclusive short-term lock from the ServerManager
    RequestSharedLock(Inst) $\rightarrow$ ServerManager;[3]
    RequestShortTermLock(Inst) $\rightarrow$ ServerManager;
    // change the set of active servers (cf. Algorithm 2)
    if LastBranch(Inst) then
        // the migration is performed for the last execution branch of the WF instance, that is active at the
        // SourceServer
        UpdateActiveServers(Inst, SourceServer, LogOff, TargetServer) $\rightarrow$ ServerManager;
    else    // another execution path is active at SourceServer
        UpdateActiveServers(Inst, SourceServer, Stay, TargetServer) $\rightarrow$ ServerManager;
```

[3] *p()* $\rightarrow s$ means that procedure p is called and then executed by server s.

```
    // perform the actual migration and release the non-exclusive lock
    MigrateWorkflowInstance(Inst) $\rightarrow$ TargetServer;
    ReleaseSharedLock(Inst) $\rightarrow$ ServerManager;
end.
```

Algorithm 2 is used by the *ServerManager* to manage the WF servers currently involved in controlling a given WF instance. To fulfill this task, the *ServerManager* also has to manage the locks mentioned above. If the procedure *UpdateActiveServers* is called with the option *LogOff*, the source WF server of the migration is deleted from the set *ActiveServers(Inst)*; i.e., the set of active WF servers with respect to the given WF instance. The reason for this is that this WF server is no longer involved in controlling this WF instance. The target WF server for the migration, however, is always inserted into this set independently of whether it is already contained or not because this operation is idempotent.

The short-term lock requested by Algorithm 1 before the invocation of *UpdateActiveServers* prevents Algorithm 2 from being run in parallel more than once for a given WF instance. This helps to avoid an error due to overlapping changes of the set *ActiveServers(Inst)*. When this set has been adapted, the short-term lock is released.

Algorithm 2 (*UpdateActiveServers*: Managing the Active WF Servers)

```
input
    Inst: ID of the affected WF instance
    SourceServer: source server of the migration
    Option: source server be involved in the WF instance furthermore (Stay) or not (LogOff)?
    TargetServer: target server of the migration
begin
    // update the set of the current WF servers of the WF instance Inst
    if Option = LogOff then
        ActiveServers(Inst) = ActiveServers(Inst) $-$ {SourceServer};
    end if
    ActiveServers(Inst) = ActiveServers(Inst) $\cup$ {TargetServer};
    ReleaseShortTermLock(Inst); // release the short-term lock
end.
```

Performing Ad-hoc Modifications. Where the previous section has described how the *ServerManager* handles the set of currently active WF servers for a particular WF instance, this section sets out how this set is utilized when ad-hoc modifications are performed.

First of all, if no parallel branches are currently being executed, trivially, the set of active WF servers contains exactly one element, namely the current WF server. This case may be detected by making use of the state and structure information (locally) available at the current WF server. The same applies to the special case that currently all parallel branches are controlled by the same WF server. In both cases, the method described in the following is not needed and therefore not applied. Instead, the WF server currently controlling the WF instance performs the ad-hoc modification without consulting any other WF server. Consequently, this WF server must not communicate with the *Server-Manager* as well. For this special case, therefore, no additional synchronization effort occurs (when compared to the central case).

We now consider the case that parallel branches exist; i.e., an ad-hoc modification of the WF instance may have to be synchronized between multiple WF servers. The WF server which coordinates the ad-hoc modification then requests the set *ActiveServers* from the *ServerManager*. When performing the ad-hoc modification, it is essential that this set is not changed due to concurrent migrations. Otherwise, wrong WF servers would be involved in the modification procedure. In addition, it is vital that the WF execution schema of the WF instance is not restructured due to concurrent modifications, since this may result in the generation of an incorrect schema.

To prevent either of these faults we introduce Algorithm 3. It requests an exclusive lock from the *ServerManager* to avoid the mentioned conflicts. This lock corresponds to a write lock [11] in a database system and is incompatible with read locks (*RequestSharedLock* in Algorithm 1) and other write locks of the same WF instance. Thus, it prevents that migrations are performed simultaneously to an ad-hoc modification of the WF instance.

Algorithm 3 (Performing an Ad-hoc Modification)

```
input
    Inst: ID of the WF instance to be modified
    Modification: specification of the ad-hoc modification
begin
    // calculate the ServerManager for this WF instance
    ServerManager = StartServer(Inst);
    // request an exclusive lock from the ServerManager and calculate the set of active WF servers
    RequestExclusiveLock(Inst) $\rightarrow$ ServerManager;
    ActiveServers = GetActiveServers(Inst) $\rightarrow$ ServerManager;
    // request a lock from all servers, calculate the current WF state, and perform the change (if possible)
    for each Server $s \in$ ActiveServers do
        RequestStateLock(Inst) $\rightarrow$ s;
    GlobalState = GetLocalState(Inst);
    for each Server $s \in$ ActiveServers do
        LocalState = GetLocalState(Inst) $\rightarrow$ s;
        GlobalState = GlobalState $\cup$ LocalState;
    if DynamicModificationPossible(Inst, GlobalState, Modification) then
        for each Server $s \in$ ActiveServers do
            PerformDynamicModification(Inst, GlobalState, Modification) $\rightarrow$ s;
    // release all locks
    for each Server $s \in$ ActiveServers do
```

```
        ReleaseStateLock(Inst) → s;
    ReleaseExclusiveLock(Inst) → ServerManager;
end.
```

As soon as the lock has been granted, a query is sent to acquire the set of active WF servers of this WF instance.[4] Then a lock is requested at all WF servers belonging to the set *ActiveServers* in order to prevent local changes to the state of the WF instance. Any activities already started, however, may be finished normally since this does not affect the applicability of an ad-hoc modification. Next the (locked) state information is retrieved from all active WF servers. Note that the resulting global and current state of the WF instance is required to check whether the ad-hoc modification to be performed is permissible or not. In Figure 3, for example, WF server s_4, which is currently controlling activity g and which wants to insert activity x after activity g and before activity d, normally does not know the current state of activity d (from the parallel branch). Yet the ad-hoc modification is permissible only if activity d has not been started at the time the modification is initiated [16]. If this is the case, the modification is performed at all active WF servers of the WF instance (*PerformDynamic-Modification*). Afterwards, the locks are released and any blocked migrations or modification procedures may then be carried out.

3.3 Illustrative Example

How migrations and ad-hoc modifications work together is explained by means of an example. Figure 4a shows a WF instance, which is currently controlled by only one WF server, namely the WF server s_1. Figure 4b shows the same WF instance after it migrated to a second WF server (s_2). In Figure 4c the execution was continued. One can also see that each of the two WF servers must not always possess complete information about the global state of the WF instance.

Assume now that an ad-hoc modification has to be performed, which is coordinated by the WF server s_1. Afterwards, both WF servers shall possess the current schema of the WF instance to correctly proceed with the flow of control. With respect to the (complete) current state of the WF instance, it is sufficient that it is known by the coordinator s_1 (since only this WF server has to decide on the applicability of the desired modification). The other WF server only carries out the modification (as specified by WF server s_1).

4 Distributed Execution of a Modified Workflow Instance

If a migration of a WF instance has to be performed, its current state has to be transmitted to the target WF server. In ADEPT, this is done by transmitting the relevant parts of the execution history of the WF instance together with the

[4] This query may be combined with the lock request into a single call to save a communication cycle.

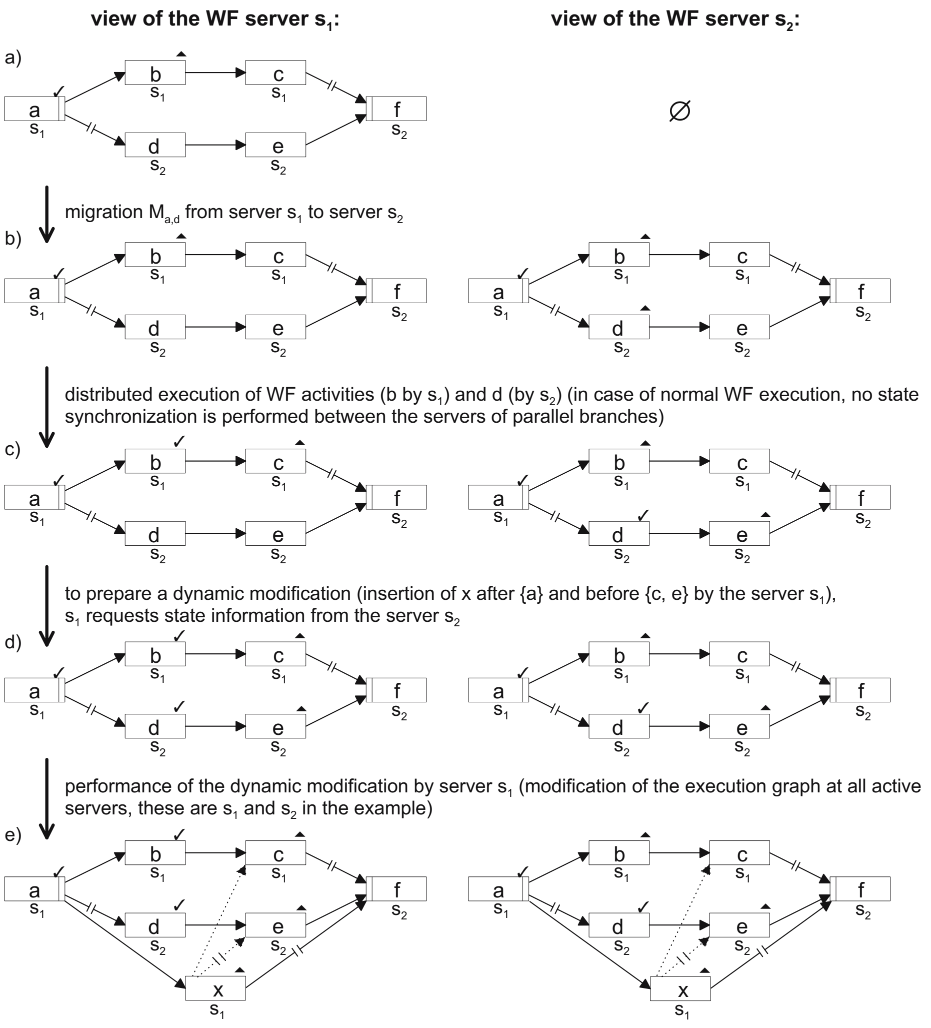

view of the WF server s1:
view of the WF server s2:
a)
∅
migration M a,d from server s1 to server s2
b)
distributed execution of WF activities (b by s1) and d (by s2) (in case of normal WF execution, no state synchronization is performed between the servers of parallel branches)
c)
to prepare a dynamic modification (insertion of x after {a} and before {c, e} by the server s1), s1 requests state information from the server s2
d)
performance of the dynamic modification by server s1 (modification of the execution graph at all active servers, these are s1 and s2 in the example)
e)
a
b
c
d
e
f
x
s1
s2

Fig. 4. Effects of migrations and ad-hoc modifications on the (distributed) execution schema of a WF instance (local view of the WF servers)

values of WF relevant data (i.e., data elements used as input and output data of WF activities or as input data for branching and loop conditions)

If an ad-hoc modification was previously performed, the target WF server of a migration also needs to know the modified execution schema of the WF instance in order to be able to control the WF instance correctly. In the approach introduced in the previous section, only the active WF servers of the WF instance to be modified have been involved in the modification. As a consequence, the WF servers of subsequent activities, however, still have to be informed about the modification. In our approach, the necessary information is transmitted upon migration of the WF instance to the WF servers in question. Since migrations are rather frequently performed in distributed WfMS, this communication needs to be performed efficiently. Therefore, in Section 4.1 we introduce a technique which fulfills this requirement to a satisfactory degree. Section 4.2 presents an enhancement of the technique that precludes redundant data transfer.

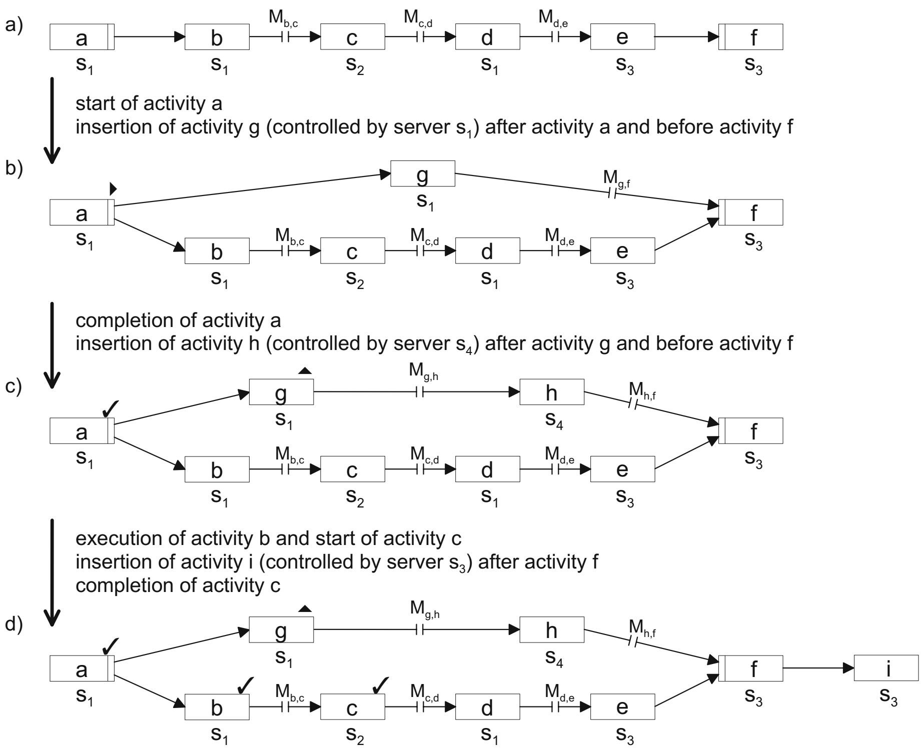
4.1 Efficient Transmission of Information About Ad-Hoc Modifications

In the following, we examine how a modified WF execution schema can be communicated to the target WF server of a migration. The key objective of this investigation is the development of an efficient technique that reduces communication-related costs as far as possible.

Of course, the simplest way to communicate the current execution schema of the respective WF instance to the migration target server is to transmit this schema in whole. Yet this technique burdens the communication system unnecessarily because related WF graph of this WF schema may comprise a large number of nodes and edges. This results in an enormous amount of data to be transferred – an inefficient and cost-intensive approach. Apart from this, the entire execution schema does not need to be transmitted to the migration target server as the related WF template has been already located there. (Note that a WF template is being deployed to all relevant WF servers before any WF instance may be created from it.) In fact, in most cases the current WF schema of the WF instance is almost identical to the WF schema associated with the WF template. Thus it is more efficient to transfer solely the relatively small amount of data which specifies the modification operation(s) applied to the WF instance. It would therefore seem practical to use the change history for this purpose. In ADEPT the migration target server needs this history anyway [16], so that its transmission does not lead to an additional effort. When the base operations recorded in the change history are applied to the original WF schema of the WF template, the result is the current WF schema of the given WF instance. This simple technique dramatically reduces the effort necessary for communication. In addition, as typically only very few modifications are performed on any individual WF instance, computation time is kept to a minimum.

4.2 Enhancing the Method Used to Transmit Modification Histories

Generally, one and the same WF server may be involved more than once in the execution of a WF instance – especially in conjunction with loops. In the example from Figure 5, for instance, WF server s_1 hands over control to WF server s_2 after completion of activity b but will receive control again later in the flow to execute activity d. Since each WF server stores the change history until being informed that the given WF instance has been completed, such a WF server s already knows the history entries for the modifications it has performed itself. In addition, s knows any modifications that had been effected by other WF servers before s handed over the control of the WF instance to another WF


a)
a s1
b s1
Mb,c
c s2
Mc,d
d s1
Md,e
e s3
f s3
start of activity a
insertion of activity g (controlled by server s1) after activity a and before activity f
b)
g s1
Mg,f
completion of activity a
insertion of activity h (controlled by server s4) after activity g and before activity f
c)
Mg,h
h s4
Mh,f
execution of activity b and start of activity c
insertion of activity i (controlled by server s3) after activity f
completion of activity c
d)
i s3

e) Start(a, s_1, ...), DynModif(1), End(a, s_1, ...), DynModif(2), Start(b, s_1, ...), End(b, s_1, ...), Start(c, s_2, ...), DynModif(3), End(c, s_2, ...)

Fig. 5. a-d) WF instance and e) Execution history of WF server s_2 after completion of activity c. – In case of distributed WF control, with each entry the execution history records the WF server responsible for the control of the corresponding activity.

server for the last time. Hence the data related to this part of the change history need not be transmitted to the WF server. This further reduces the amount of data required for the migration of the "current execution schema".

Transmitting Change History Entries. An obvious solution for avoiding redundant transfer of change history entries would be as follows: The migration source server determines from the existing execution history exactly which modification the target WF server must already know. The related entries are then simply not transmitted when migrating the WF instance. In the example given in Figure 5, WF server s_2 can determine, upon ending activity c, that the migration target server s_1 must already know the modifications 1 and 2. In the execution history (cf. Figure 5e), references to these modifications (*DynModif(1)* and *DynModif(2)*) have been recorded before the entry *End(b, s_1, ...)* (which was logged when completing activity b). As this activity was controlled by WF server s_1, this WF server does already know the modifications 1 and 2. Thus, for the migration $M_{c,d}$, only the change history entry corresponding to modification 3 needs to be transmitted. The transmitted part of the change history is concatenated with the part already present at the target server before this WF server generates the new execution schema and proceeds with the flow of control.

In some cases, however, redundant transfer of change history data cannot be avoided with this approach: As an example take the migrations $M_{d,e}$ and $M_{h,f}$ to the WF server s_3. For both migrations, with the above approach, all entries corresponding to modifications 1, 2, and 3 must be transmitted because the WF server s_3 was not involved in executing the WF instance thus far. The problem is that the migration source servers s_1 and s_4 are not able, from their locally available history data, to derive whether the other migration from the parallel branch has already been effected or not. For this reason, the entire change history must be transmitted. Yet with the more advanced approach set out in the next section, we can avoid such redundant data transfer.

Requesting Change History Entries. To avoid redundant data transmissions as described in the previous section, we now sketch a more sophisticated method. With this method, the necessary change history entries are explicitly requested by the migration target server. When a migration takes place, the target WF server informs the source WF server about the history entries it already knows. The source WF server then only transmits those change history entries of the respective WF instance which are yet missing on the target server. In ADEPT, a similar method has been used for transmitting execution histories; i.e., necessary data is provided on basis of a request from the migration target server. Here, no additional effort is expended for communication, since both, the request for and the transmission of change history entries may be carried out within the same communication cycle.

With the described method, requesting the missing part of a change history is efficient and easy to implement in our approach. If the migration target server was previously involved in the control of the WF instance, it already possesses all entries of the change history up to a certain point (i.e., it knows all ad-hoc modifications that had been performed before this server handed over control the last time). But from this point on, it does not know any further entries. It is thus sufficient to transfer the ID of the last known entry to the migration source server to specify the required change history entries. The source WF server then transmits all change history entries made after this point. Due to lack of space we omit further details.

To sum up, with our approach not only ad-hoc modifications can be performed efficiently in a distributed WfMS (see Section 3), transmission costs for migration of modified WF instances may also be kept very low.

5 Related Work

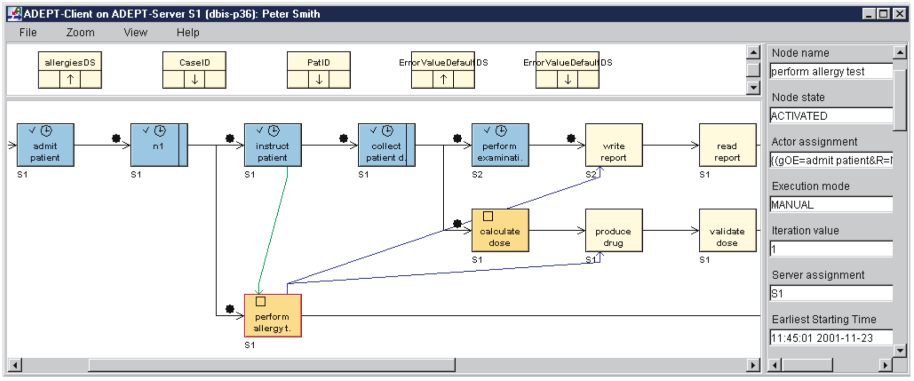
There are only few approaches which address both WF modification issues and distributed WF control [8,9,13,21,2]. WIDE [9] allows WF schema modifications and their propagation to running WF instances (if compliant to the new schema). In addition, control of WF instances is distributed [9]. Thereby, the set of the potential actors of an activity determines the WF server which is to control this activity. In MOKASSIN [13] and WASA [20,21], distributed WF execution is realized through an underlying CORBA infrastructure. Both approaches do not discuss the criteria used to determine a concrete distribution of the tasks; i.e., the question which WF server has to control a specific activity remains open. Here, modifications may be made at both, the WF schema and the WF instance level under consideration of correctness issues. INCAs [2] realizes WF instance control by means of rules. WF control is distributed, in INCAs, with a given WF instance controlled by that processing station that belongs to the actor of the current activity. The mentioned rules are used to calculate the processing station of the subsequent activity and, thereby, the actor of that activity. With this approach, it is possible to modify the rules, what results in an ad-hoc change of the WF instance behavior. As opposed to the approach presented in this paper, all these approaches do not explicitly address how ad-hoc modifications and distributed WF execution interact. The approach proposed in [10] enables some kind of flexibility in distributed WfMS as well, especially in the context of virtual enterprises. However, it does not allow to modifiy the structure of in-progress WF instances. Instead, the activities of a WF template represent placeholders for which the concrete implementations are selected at run-time.

In the WF literature, some approaches for distributed WF management are cited where a WF instance is controlled by one and the same WF server over its entire lifetime; e.g., Exotica [1] and MOBILE [12]. (The latter approach was extended in [18] that way that a sub-process may be controlled by a different WF server, which is determined at run-time.) Although migrations are not performed, different WF instances may be controlled by different WF servers. And, since a central control instance exists for each WF instance in these approaches, ad-hoc modifications may be performed just as in a central WfMS. Yet there is a drawback with respect to communication costs: The distribution model does not allow to select the most favorable WF server for the individual activities. When developing ADEPT, we therefore did not follow such an approach since the additional costs incurred in standard WF execution are higher than the savings generated due to the (relatively seldom performed) ad-hoc modifications.

6 Summary

Both distributed WF execution and ad-hoc modification are essential functions of any WfMS. Each of these aspects is closely linked with a number of requirements and objectives that are, to some extent, opposing. Reason for this is that the central control instance necessary for ad-hoc modifications typically impacts the efficiency of distributed WF execution. Therefore, we cannot afford to consider these two aspects separately. An investigation of exactly how these functions interact has been presented. And the results show that they are, in fact, compatible: We have realized ad-hoc modifications in a distributed WfMS.

Our approach also allows efficient distributed control of previously modified WF instances since only a part of the relatively small change history needs to be transmitted when transferring the modified execution schema. This is vital as migrations are frequent. To conclude, ADEPT succeeds in seamlessly integrating both distributed WF execution and ad-hoc WF modifications into a single system. The presented concepts have been implemented in a powerful proof-of-concept prototype, which constitutes the distributed variant of the ADEPT system (cf. Fig. 6). It shows that one can really build a WfMS which offers the described functionality within one system (for details see [7]). It also shows, however, that such a high-end WfMS is a large software systems, easily reaching the code complexity of high-end database management systems.



Node name
perform allergy test
Node state
ACTIVATED
Actor assignment
{(gOE=admit patient&R=I
Execution mode
MANUAL
Iteration value
1
Server assignment
S1
Earliest Starting Time
11:45:01 2001-11-23

Fig. 6. ADEPT monitoring component showing a distributed workflow controlled by servers S1 and S2 after its runtime modification

References

1. Alonso, G., Kamath, M., Agrawal, D., El Abbadi, A., Günthör, R., Mohan, C.: Failure Handling in Large Scale Workflow Management Systems. Technical Report RJ9913, IBM Almaden Research Center (1994)
2. Barbará, D., Mehrotra, S., Rusinkiewicz, M.: INCAs: Managing Dynamic Workflows in Distributed Environments. J. of Database Management 7(1), 5–15 (1996)
3. Bauer, T., Dadam, P.: A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration. In: Proc. CoopIS 1997, Kiawah Island, SC, pp. 99–108 (1997)
4. Bauer, T., Dadam, P.: Distribution Models for Workflow Management Systems. Informatik Forschung und Entwicklung 14(4), 203–217 (1999) (in German)
5. Bauer, T., Dadam, P.: Efficient Distributed Workflow Management Based on Variable Server Assignments. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 94–109. Springer, Heidelberg (2000)
6. Bauer, T., Reichert, M., Dadam, P.: Intra-Subnet Load Balancing for Distributed Workflow Management Systems. Int. J. Coop. Inf. Sys. 12(3), 295–323 (2003)

7. Bauer, Th., Reichert, M.: An Approach for Supporting Ad-hoc Process Changes in Distributed Workflow Management Systems. Technical report, University of Twente, CTIT (September 2007)
8. Cao, J., Yang, J., Chan, W., Xu, C.: Exception handling in distributed workflow systems using mobile agents. In: Proc. ICEBE 2005, pp. 48–55 (2005)
9. Casati, F., Grefen, P., Pernici, B., Pozzi, G., Sánchez, G.: WIDE: Workflow Model and Architecture. CTIT Technical Report 96-19, University of Twente (1996)
10. Cichocki, A., Georgakopoulos, D., Rusinkiewicz, M.: Workflow Migration Supporting Virtual Enterprises. In: Proc. BIS 2000, Poznań, pp. 20–35 (2000)
11. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, San Francisco (1993)
12. Jablonski, S.: Architecture of Workflow Management Systems. Informatik Forschung und Entwicklung 12(2), 72–81 (1997) (in German)
13. Joeris, G., Herzog, O.: Managing Evolving Workflow Specifications. In: Proc. CoopIS 1998, New York, pp. 310–321 (1998)
14. Lenz, R., Reichert, M.: IT Support for Healthcare Processes - Premises, Challenges, Perspectives. DKE 61, 82–111 (2007)
15. Muth, P., Wodtke, D., Weißenfels, J., Kotz-Dittrich, A., Weikum, G.: From Centralized Workflow Specification to Distributed Workflow Execution. JIIS 10(2), 159–184 (1998)
16. Reichert, M., Dadam, P.: ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. JIIS 10(2), 93–129 (1998)
17. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases 16(1), 91–116 (2004)
18. Schuster, H., Neeb, J., Schamburger, R.: A Configuration Management Approach for Large Workflow Management Systems. In: Proc. Int. Conf. on Work Activities Coordination and Collaboration, San Francisco (1999)
19. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: CAiSE 2007. Proc. 19th Int'l Conf. on Advanced Information Systems Engineering, pp. 574–588 (2007)
20. Weske, M.: Flexible Modeling and Execution of Workflow Activities. In: Proc. 31st Hawaii Int. Conf. on Sys Sciences, Hawaii, pp. 713–722 (1998)
21. Weske, M.: Workflow Management Through Distributed and Persistent CORBA Workflow Objects. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 446–450. Springer, Heidelberg (1999)

Acquaintance Based Consistency in an Instance-Mapped P2P Data Sharing System During Transaction Processing

Md Mehedi Masud and Iluju Kiringa

SITE, University of Ottawa, Canada
{mmasud,kiringa}@site.uottawa.ca

Abstract. The paper presents a transaction processing mechanism in a peer-to-peer (P2P) database environment that combines both P2P and database management systems functionalities. We assume that each peer has an independently created relational database and data heterogeneity between two peers is resolved by data-level mappings. For such an environment, the paper first introduces the execution semantics of a transaction and shows the challenges for concurrent execution of transactions, initiated from a peer, over the network. Later the paper presents a correctness criterion that ensures the correct execution of transactions over the P2P network. We present two approaches ensuring the correctness criterion and finally discuss the implementation issues.

1 Introduction

In the past few years peer-to-peer (P2P) technology has emerged as a new paradigm for distributed data sharing systems. In P2P all participating computers (or peers) have equivalent capabilities and responsibilities and exchange resources and services through pair-wise communication by eliminating the need for centralized servers. Until now there are many domain specific P2P systems (e.g. Freenet, Gnutella, SETI@home, ICQ, etc.) have already been deployed. With a few notable exceptions, currently implemented P2P systems lack data management capabilities that are typically found in database management system (DBMS).

A P2P database system (P2PDBS) combines both P2P and database management systems functionalities. In a P2PDBS, a peer provides access to its resources to other peers and shares its data with other peers through the pair-wise communication. A P2PDBS is similar to a conventional multidatabase system (MDBS) in the sense that each system consists of a collection of independently created local database systems (LDBSs), and transaction management is handled at both the global and local levels. In a MDBS, global level transactions are issued to the global transaction manager (GTM), where they are decomposed into a set of global subtransactions to be individually submitted to the corresponding LDBSs. Local transactions are directly submitted to the local transaction management systems (LTMs). Each local transaction manager maintains the correct execution of both local transactions and global subtransactions at its site. It is left to the GTM to maintain the correct execution of global transactions.

In contrast, a P2PDBS is built on a dynamic network of peers without a global transaction manager or controller. In a P2PDBS, global level transactions are initiated by any peer. If a transaction is submitted to a peer and needs to be executed over the network, then the transaction is propagated from peer to peer. Note that when a user submits a transaction to a peer, he/she is only aware of the local database schema. The mappings between the peer, where the transaction is active, and other peers in the network determine the translation of the transaction and propagation and execution of the transaction to other peers. The logical connection, that is established through mappings, between two peers is called an acquaintance. The acquaintance is established either with data-level mappings [1] or schema-level mappings [3]. In this paper, we use data-level mappings created from mapping tables [1] to establish an acquaintance. Intuitively, mapping tables provide data-level mappings which list pairs of corresponding values between two sources. The use of mapping tables does not require peers to disclose their schemas and mappings can be established between peers that belong to different worlds but store related data. Basically, mapping tables act as an interface or as a data descriptor to relate data between two peers. The domain of such settings can be applied to biological, health care, flight reservation, and business to business databases systems.

One of the prime objectives of transaction management is to guarantee serializable execution of local and global transactions. In a conventional distributed database system, serializability is ensured using the *distributed two-phase locking* (2PL) protocol and atomicity of transactions is ensured using the *two-phase commit* (2PC) protocol. Several solutions have been proposed for ensuring serializable execution of global transactions in MDBSs [4,6,9,7]. Such solutions are usually applied to short living transactions and for systems where all local databases can be logically integrated through a dedicated central software module. However, they are not well suited for a P2P environment because such a centralized component can't be created due to the dynamic nature of peers and the arbitrary topology of P2P networks.

1.1 Objectives, Assumptions, and Contributions

Although we assume that each LDBS of each peer guarantees serializability, concurrent global transactions that execute in multiple peers may be serialized in different orders at different peers resulting in a non-serializable global schedule. This paper identifies some potential problems ensuring global serializability during concurrent execution of global transactions in a P2PDBS and presents two approaches that ensure global serializability overcoming the problems. The main feature of the approaches is that we do not need any change to the underlying LDBSs. The model discussed in this paper is based on the following assumptions:

1. When a user submits a transaction to a peer, he/she is only aware of the local database schema and there is no global transaction manager or coordinator in the system and no changes are allowed to the local database system software.

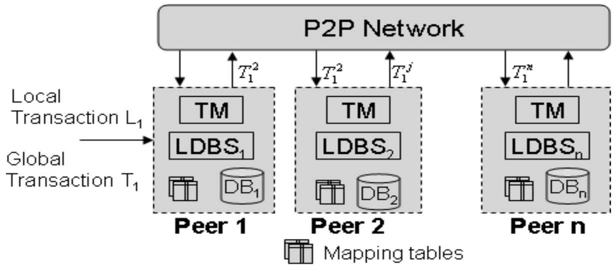


Fig. 1. The P2PDBS model

2. An LDBS is not allowed to distinguish between a local and global transaction which are active at the local peer.
3. A peer can communicate with another peer only for sending and receiving transaction messages and acknowledgements. However, a peer is not able to communicate with a peer to synchronize the execution of global transactions.
4. Each LDBS uses its own locking protocol and has a mechanism for ensuring local serializability.

Our contributions are as follows:

– We introduce a transaction execution semantics in a P2PDBS where databases may contain related data that overlap little, if at all. The semantics relies on the translation of transactions between peers through the use of mapping tables.
– We analyze the execution semantics of transactions initiated by a peer and identify the potential problems ensuring global serializable execution of the transactions.
– We introduce a correctness criterion for the consistency of a P2PDBS during execution of transactions and propose two approaches for ensuring correctness criterion without violating the autonomy of LDBSs.
– Finally, we briefly discuss the implementation issues.

2 P2P Database Systems Model

A P2PDBS is a set $P = \{P_1, P_2, \cdots, P_n\}$ of n peers with autonomous preexisting local database systems (LDBSs). Formally, each peer P_i $(1 \leq i \leq n)$ is defined by a pair of $< DB_i, M_i >$, where DB_i is a database and M_i is a set of mapping tables. A set of mapping tables $M_{ij} = \{M_{ij}^1, \cdots, M_{ij}^k\} \subseteq M_i$ in P_i stores the data mappings between P_i and P_j that P_i shares its local data with P_j. The placement of mapping tables M_{ij} creates an acquaintance $(i \rightarrow j)$ between P_i and P_j. Here, P_j is called the *acquaintee* of P_i. Each peer provides transaction service (local and remote) to local and remote users in order to access its database. To illustrate, a P2PDBS model is shown in Figure 1. The figure shows that peers are connected in a P2P network and each peer is attached with a database and a set of mapping tables. We see that both the local and global transactions can be initiated in a peer. The system has no centralized controller. Acquaintances are established between peers by mapping tables.

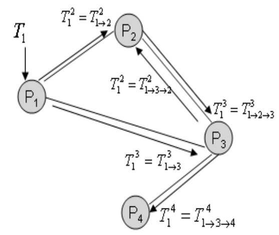
For the purpose of this paper, we use the well-known read-write model of transactions. We now recall the basics of this model. Let a database be a (finite) set $D = \{a, b, c, \cdots\}$ of data objects. A transaction T is a partially ordered set of database operations applied to data object $a \in D$. Formally, T=$(O_T, \prec_T)$, where O_T, is a finite set of operations and $\prec_T$ is a partial order operations that can be invoked by a transaction T. The operations of a transaction T consists of a set of *read* (denoted by $r(a)$) and *write* (denoted by $w(a)$) operations. Further, each T has begin and termination operations commit or abort. For simplicity, we do not consider the begin and termination operations, but assume that all transactions under consideration end successfully.

The concurrent execution of transactions results in a schedule. A schedule S=$(\varGamma_S, \prec_S)$, where $\varGamma_S$ is a finite set of transactions and $\prec_S$ is a partial order over the operations belonging to transactions in $\varGamma_S$. The partial order $\prec_S$ satisfies the property that it preserves the order of steps within each transaction (that is, $\prec_{T_i} \subseteq \prec_S$, for each $T_i \in \varGamma_S$).

A transaction T_i is said to conflict (direct conflict) with T_j, denoted by $T_i \rightarrow^c T_j$, if there exist operations o_i in T_i and o_j in T_j, $T_i \neq T_j$, such that $o_i \prec_S o_j$, and o_i, o_j access the same data item and one of them is a write operation. By $\overset{*}{\rightarrow}^c$ we denote the transitive closure (indirect conflict) of the $\rightarrow^c$ relation. Another way we can say that a transaction T_i is said to indirectly conflict with T_j, denoted by $T_i \overset{*}{\rightarrow}^c T_j$, if there exist operations o_k in T_k such that T_i conflicts with T_k and T_k conflicts with T_j. In this paper, we call the conflict relation between transactions as serialization order.

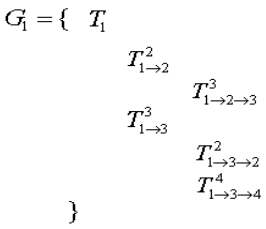
In a P2PDBS, a transaction T_i initiates and executes locally at a peer P_i in which case the appropriate updates are done on the local DB_i. Our thesis is that for a user to issue a transaction, he/she needs only be aware of the local schema she is using. T_i may execute in other peers $P_j (1 \leq j \leq n)$ in a P2PDBS subject to the mapping constraint between P_i and P_j. Hence, if T_i is propagated to P_j from P_i then there must exist data mappings between P_i and P_j with the accessed data by T_i. Thus, all transactions in a P2PDBS are submitted locally while they may be processed globally in the P2P network. When a transaction T_i needs to execute globally, the local peer translates the transaction for each of its acquaintee which is relevant to T_i. A peer P_j is relevant to T_i, if T_i can be translated for P_j over the acquaintance $(i \rightarrow j)$ with respect to the mapping tables M_{ij}. The translation is denoted as $T_i \xrightarrow{M_{ij}} T_j$. Each transaction is defined, in terms of syntax, with respect to the schema of local DB_i. Based on the execution semantics, transactions are classified into three categories, namely *local*, *remote*, and *global* transactions.

-**Local transaction** (L_i): A transaction L_i submitted at a peer P_i and accessing only the local database DB_i.

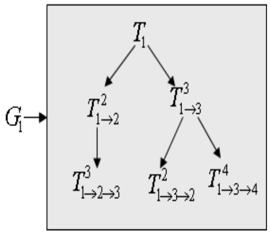


T_1
$T_1^2 = T_{1\to2}^2$
$T_1^2 = T_{1\to3\to2}^2$
$T_1^3 = T_{1\to2\to3}^3$
$T_1^3 = T_{1\to3}^3$
$T_1^4 = T_{1\to3\to4}^4$
P1
P2
P3
P4

(a) Global transaction propagation in a network



(b) Global transaction structure (Layered view)



(c) Global transaction structure (Tree view)

Fig. 2. Global transaction structure

-**Remote transaction** $(T_{i\to j}^j)$: A transaction $T_{i\to j}^j$ is a remote transaction at P_j accessing database DB_j that is originated at P_i. The subscript $(i \to j)$ denotes the propagation path of T_i from P_i to P_j through which T_i has been propagated. For the sake of presentation, we sometimes omit the propagation path from the notation of a *remote* transaction. For example, we will represent the remote transaction $T_{i\to j}^j$ as T_i^j.

-**Global transaction** (G_i): A *global* transaction G_i=$\{T_i, T_i^j, T_i^k, \cdots\}$ consists of T_i originated at P_i and a set of *remote* transactions $T_i^j (1 \leq j \leq n)$.

Note: For ease of presentation we sometimes denote $T_i^j, T_i^k, \cdots$ as T_i since they are actually generated from T_i and a global transaction G_i is represented with the initiator T_i. Intuitively, execution of any component transaction $T_i, T_i^j, T_i^k, \cdots$ is called the execution of G_i.

2.1 Global Transaction Structure

We now examine the structure of a global transaction G_i created from a transaction T_i at P_i. When P_i generates a set of remote transactions from T_i, using mappings, to be executed in its immediate acquaintees, G_i can be viewed as a two-level transaction. In this case, T_i becomes the root of G_i. G_i becomes a multi-level transaction when the acquaintees of P_i also generate remote transactions for their respective acquaintees. Consequently, a global transaction may have multiple layers depending on the number of hops it propagates.

Intuitively, as transactions are propagated in the system acquaintance-by-acquaintance, a (directed) transaction dependency tree is induced. The nodes in this tree represent transactions and there is an edge from T_i^j to T_i^k, if P_j forwards T_i to P_k for execution. For each edge we define the relationship as parent-child relationship. Therefore, $T_i^j (T_i^k)$ is the parent (child) of $T_i^k (T_i^j)$ and denoted as $T_i^j \mathcal{P} T_i^k (T_i^k \mathcal{C} T_i^j)$

Example 1. Consider the P2P network of Figure 2(a). The network consists of four peers P=(P_1, P_2, P_3, P_4). The undirected edges between peers represent the acquaintances between them and the directed edges depict transaction propagations. We assume that the acquaintances are bi-directional. Consider the construction of a global transaction G_1 initiated by T_1 at P_1. Assume that T_1 needs to be executed at P_2 and P_3 due to the data mappings for the data items mentioned by T_1. Therefore, P_1 translates T_1 for P_2 and P_3 and forwards the translated transactions T_1^2 and T_1^3 to P_2 and P_3 respectively. After execution, P_2 also forwards T_1^2 to P_4 and P_3 and peer P_3 forwards T_1^3 to P_2. Therefore, the global transaction generated from T_1 is $G_1 = \{T_1, T_1^2, T_1^3, T_1^4\}$. Note that, both P_2 and P_3 forward T_1 to each other. We omitted these transactions, because they previously received T_1 from P_1. Figure 2(b) and 2(c) show the layered and tree view of G_1

2.2 Transaction Translation

When a transaction is forwarded to an acquaintee for execution, first the transaction is translated in terms of the data vocabularies of the acquaintee. During translation, each $r(a)$ and $w(a)$ operation is translated into the same operation substituting the data value a with the corresponding data value a' using the appropriate mapping table that contains the association $a \rightarrow a'$ and the translation restriction is followed which is defined below.

Definition 1 (Translation Restriction). *Consider two transactions T_i and T_j with partial orders $\prec_{T_j}$ and $\prec_{T_j}$ respectively. Transaction T_j follows translation restriction of T_i if $O_{T_j} \subseteq O_{T_i}$ and for all o_1,$o_2 \in O_{T_j}$, $o_1 \prec_{O_{T_j}} o_2$ iff $o_1 \prec_{O_{T_i}} o_2$.*

Note that the translation keeps each operation (read/write) same but only translates the data items mentioned in the operations.

Definition 2 (Schedule Translation Restriction). *For a schedule $S_i = (\Gamma_{S_i}, \prec_{S_i})$ we define a schedule $S_j = (\Gamma_{S_j}, \prec_{S_j})$ follows the translation restriction of S_i if for operations o_1,o_2 in S_j, $o_1 \prec_{S_j} o_2$, iff $o_1 \prec_{S_i} o_2$.*

Translation of a transaction may be either partial or complete over the acquaintance.

Definition 3 (Partial Translation). *A transaction T_j is a partial translation of a transaction T_i, if $O_{T_j} \subset O_{T_i}$ and T_j follows the translation restriction of T_i.*

Definition 4 (Complete Translation). *A transaction T_j is a complete translation of a transaction T_i, if $O_{T_j} = O_{T_i}$ and T_j follows the translation restriction of T_i.*

Example 2. Let a global transaction G_1 is initiated by T_1 at P_1 in the P2P network of Figure 2(a).

$$T_1 = w_1(a^1) r_1(b^1) w_1(c^1) w_1(d^1)$$

Suppose the following data mappings exist in different acquaintances.

$$(1 \rightarrow 2)\text{: } a^1 \rightarrow a^2,\ c^1 \rightarrow c^2,\ d^1 \rightarrow d^2$$
$$(1 \rightarrow 3)\text{: } c^1 \rightarrow c^3,\ d^1 \rightarrow d^3, \quad (3 \rightarrow 4)\text{: } c^3 \rightarrow c^4,\ d^3 \rightarrow d^4$$

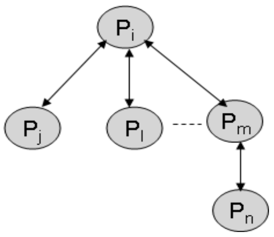


Fig. 3. The peer P_i and its acquaintees

Based on the mappings, P_1 generates the following remote transactions from T_1 for P_2 and P_3.

$$T_1^2{=}w_1(a^2)w_1(c^2)w_1(d^2),\ \ T_1^3{=}w_1(c^3)w_1(d^3)$$

When P_3 receives T_1^3, it also generates the following remote transaction for P_4 using the data mappings exist between P_3 and P_4.

$$T_1^4{=}w_1(c^4)w_1(d^4)$$

From the above translation, we can say that T_1^2 and T_1^3 are partial translation of T_1. Meanwhile, T_1^4 is a complete translation of T_1^3. All of the translations also follow the translation restriction.

2.3 Transaction Dependency

In a P2PDBS, a global transaction consists of a set of transactions that includes a global transaction initiator and a set of remote transactions. Each of these transactions is called a component transaction of the global transaction. Note that each component transaction is an atomic transaction resulted from the partial or complete translation of another component transaction. Each component transaction accesses data items that are located in the peer where the transaction is active. Unlike a global transaction in a MDBS, a component transaction is not decomposed in to subtransactions to access data at acquaintees. In order to access data at acquaintees, the component transaction is propagated as an atomic transaction after translation (partial/complete) to each of the acquaintees, if there are data mappings between the acquainted peers with respect to the accessed data by the transaction. Therefore, a parent-child relationship can be considered between the component transactions considering the initiator as the root of the global transaction.

3 Consistency of a P2PDBS

In a P2PDBS, global transactions propagate from peer to peer along the acquaintances. Therefore, the consistency of a P2PDBS can be achieved by recursively ensuring the consistency of each acquaintance that is included in the propagation paths of the global transactions.

Consider Figure 3 where P_i is acquainted with P_j,P_l,$\cdots$,P_m. Peer P_m is also acquainted with P_n. In order to ensure the consistency of the P2PDBS during execution of transactions generated from P_i, it is necessary to ensure the consistency of each acquaintance of the propagation paths starting from P_i. The ultimate goal of the *acquaintance-level* consistency (AC) carries two implications:

1. All the operations of a transaction must be executed in the same order in peers P_i and P_j of an acquaintance $(i \rightarrow j)$. Formally, for all acquaintances $(j \rightarrow k)$ between P_j and P_k $(1 \leq k \leq m)$ where $T_i^j \mathcal{P} T_i^k$ and for all operations o_1,$o_2 \in O_{T_i^k}$, $o_1 \prec_{T_i^k} o_2$, iff $o_1 \prec_{O_{T_i^j}} o_2$.
2. For any concurrent execution of global transactions, it is required to maintain the consistent execution order of the transactions over all the acquaintances in the propagation paths of the global transactions. Formally, for any acquaintance $(j \rightarrow k)$ between P_j and P_k, if there are schedules $S_j = (\Gamma_{S_j}, \prec_{S_j})$ and $S_k = (\Gamma_{S_k}, \prec_{S_k})$ in P_j and P_k respectively such that each transaction in Γ_{S_k} is a translation of a transaction in Γ_{S_j}, then for all operations o_1,o_2 $\in S_k$, $o_1 \prec_{S_k} o_2$ iff $o_1 \prec_{S_j} o_2$

The first condition simply enforces the same execution order of operations of a transaction at both peers in an acquaintance. The condition can be satisfied easily by forwarding each translated transaction as a single message to the acquaintees. Each acquaintee processes the transaction just like it processes its local transactions. Therefore, the order of the operations of a single transaction is maintained. In order to meet the second condition, we need a serializability theory that ensures the same serial execution of transactions in each acquaintance of a peer. Note that the second condition cannot be fulfilled by sending the transactions serially according to the local serialization order of the sender, since the sender has no knowledge about the execution order of transactions in a remote peer. In a remote peer, a local transaction can create indirect conflict that may serialize the transactions differently. In the following, we describe some of the problems that we encounter during concurrent execution of global transactions over acquaintees. Later we present a correctness criteria that ensures the consistent execution of global transactions in an acquaintance.

4 Transactions Scheduling Problem in a P2PDBS

Generally, in a multidatabase environment, the GTM has the control over the execution of global transactions and the operations they issue. The GTM can ensure the global serializability by a direct or indirect control of the global transactions. For example, altruistic locking [4], 2PC Agent Method [8], site graph [11], and Ticket Method [10]. All of the methods have a global transaction manager which plays an important role to ensure the global serializability. However, in a P2PDBS there is no such GTM. The only assumption we can make that

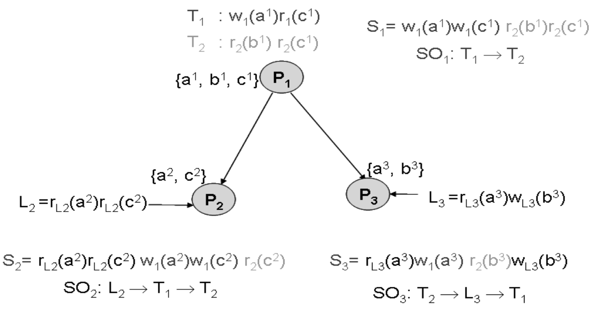


Fig. 4. Direct conflict

each peer ensures the local serializability. Once the transactions are sent to the acquaintees, the sender has no control of the execution of transactions at the acquaintees. In the following, we describe some of the cases that cause problems maintaining the global serializability when transactions are executed in the system.

Example 3 (Direct conflict). Consider a P2PDBS with three peers shown in Figure 4. Assume that peer P_1 has data items $\{a^1, b^1, c^1\}$, P_2 has data items $\{a^2, c^2\}$, and P_3 has data items $\{a^3, b^3\}$. Suppose that the transactions T_1 and T_2 are executed at P_1 concurrently and produced the schedule S_1 as follows:.

$$T_1 : w_1(a^1)w_1(c^1),\ T_2 : r_2(b^1)r_2(c^1) \quad S_1 = w_1(a^1)w_1(c^1)r_2(b^1)r_2(c^1)$$

Suppose the following data mappings exist in the acquaintances.

$$(1 \rightarrow 2)\text{: } a^1 \rightarrow a^2,\ c^1 \rightarrow c^2,\ \ (1 \rightarrow 3)\text{: } a^1 \rightarrow a^3,\ b^1 \rightarrow b^3.$$

Therefore, P_1 should translate T_1 and T_2 and forward to P_2 and P_3 for execution. The translation of T_1 and T_2 for P_2 and P_3 are as follows:

$$(P_2)\text{:}T_1{=}w_1(a^2)w_1(c^2),\ T_2{=}r_2(c^2),\ \ (P_3)\text{:}T_1{=}w_1(a^3),\ T_2{=}r_2(b^3)$$

For ease of presentation we keep the same notation of T_1 and T_2 and their translation for P_2 and P_3. Also assume that the following local transactions execute at the same time when P_1 and P_2 receive T_1 and T_2.

$$(P_2)\ \text{:}L_2 = r_{L2}(a^2)r_{L2}(c^2), \quad (P_3)\ \text{:}L_3 = r_{L3}(a^3)w_{L3}(b^3)$$

Consider that the following schedules result at P_2 and P_3.

$$S_2 = r_{L2}(a^2)r_{L2}(c^2)w_1(a^2)w_1(c^2)r_2(c^2),\ S_3 = r_{L3}(a^3)w_1(a^3)r_2(b^3)w_{L3}(b^3)$$

The resulting serialization orders at P_1, P_2, and P_3 are as follows:

$$SO_1 : T_1 \rightarrow T_2, \quad SO_2 : L_2 \rightarrow T_1 \rightarrow T_2, \quad SO_3 : T_2 \rightarrow L_3 \rightarrow T_1$$

We notice that each local schedule is serializable but they are not globally serializable with respect to S_1 of P_1 even T_1 and T_2 has direct conflict ($T_1 \rightarrow^c T_2$) at P_1. At P_3, the local transaction L_3 creates an indirect conflict between T_1 and T_2 ($T_2 \overset{*}{\rightarrow}{}^c T_1$), which serializes T_1 and T_2 in different order.

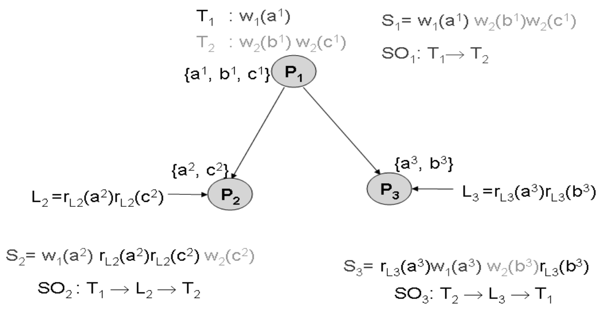


Fig. 5. Indirect conflict

Example 4 (Indirect conflict). Consider another example that the global serializability is violated even there is no conflict between transactions when they were initiated at a peer. The situation occurs when local transactions at acquaintees create indirect conflicts between the transactions. Assume that the transactions T_1 and T_2 executed concurrently at P_1 and produced the schedule S_1.

$$T_1 : w_1(a^1),\, T_2 : w_2(b^1)w_2(c^1) \quad S_1 = w_1(a^1)w_2(b^1)w_2(c^1)$$

According to the data mappings, P_1 generates the following transactions for P_2 and P_3 respectively.

$$(P_2) :T_1{=}w_1(a^2),\, T_2{=}w_2(c^2), \;\; (P_3) :T_1{=}w_1(a^3),\, T_2{=}w_2(b^3)$$

Also assume that the following local transactions execute at the same time when P_2 and P_3 receive T_1 and T_2:

$$(P_2) :L_2 = r_{L2}(a^2)r_{L2}(c^2),\, (P_3) :L_3 = r_{L3}(a^3)r_{L3}(b^3)$$

Consider that the following schedules result at P_2 and P_3 respectively.

$$S_2 = w_1(a^2)r_{L2}(a^2)r_{L2}(c^2)w_2(c^2), \quad S_3 = r_{L3}(a^3)w_1(a^3)w_2(b^3)r_{L3}(b^3)$$

We notice that T_1 and T_2 has no conflict at P_1 when they were executed and the serialization order (according to execution order) of T_1 and T_2 at P_1 is $T_1 \rightarrow T_2$. Meanwhile, the following serialization orders result at P_2 and P_3 .

$$SO_2 : T_1 \rightarrow L_2 \rightarrow T_2, \quad SO_3 : T_2 \rightarrow L_3 \rightarrow T_1$$

Notice that the serialization orders at P_2 and P_3 are different. Hence, the global serializability is violated.

5 P2PDBS Serializability Theory

When a set of global transactions $T = \{T_1, T_2, \cdots, T_n\}$ are executed concurrently in a peer P_i, the local concurrency control system of LDBS$_i$ generates a schedule S_i. The set T may be propagated over the acquaintances $(i \rightarrow j)$ between P_i and

P_j $(1 \leq j \leq m)$ and each P_j independently generates its own schedule S_j. The schedule S_i is called the parent schedule and each S_j is called the child schedule. The set of schedules $\mathbb{S}_i{=}S_i \cup (\bigcup_{j=1}^{m} S_j)$ is called the *acquaintance-level* schedule with respect to S_i at P_i.

Definition 5 (Acquaintance-Level Schedule). *An acquaintance-level schedule $\mathbb{S}_i$ with respect to S_i at P_i, is a set $\{S_i, S_1, S_2, \cdots, S_m\}$ of local schedules, where S_i is the parent schedule of a set of global transactions T and each S_j is a child schedule for each $(i \rightarrow j)$ $(1 \leq j \leq m)$.*

Definition 6 (Global P2P Schedule). *A global P2P schedule $\mathbb{S}{=}\mathbb{S}_i{\cup}(\bigcup_{j=1}^{n} \mathbb{S}_j)$ over a set of global transactions $T = \{T_1, T_2, \cdots, T_n\}$ initiated at P_i, consist of the acquaintance-level schedule $\mathbb{S}_i$ w.r.t S_i at P_i and all the acquaintance-level schedules $\mathbb{S}_j$ w.r.t S_j at P_j, $(1 \leq j \leq n, i \neq j)$ in a P2PDBS where T is executed.*

We already mentioned that in a P2PDBS, the consistency can be achieved by ensuring the consistency over each acquaintance recursively in the propagation paths of global transactions. Now, let us introduce the notion of *acquaintance-level* serial schedule.

Definition 7 (Acquaintance-Level Serial Schedule). *A global P2P schedule $\mathbb{S}$ is called acquaintance-level serial with respect to a schedule S_i and all schedules S_j of each $(i \rightarrow j)$ over a set of global transactions $T = \{T_1, T_2, \cdots, T_n\}$ if*

1. *all the local schedules in $\mathbb{S}_i$ are serializable and*
2. *for any two global transactions T_1 and T_2 in S_i, if there exist a serializable order (SO) $T_1 \rightarrow T_2$, then for all schedules $S_j \in \mathbb{S}_i(i \neq j)$, the SO is consistent between T_1 and T_2*

Theorem 1. *A global P2P schedule $\mathbb{S}$ is acquaintance-level serializable between peers P_i and all acquaintees P_j of P_i $(1 \leq j \leq m)$, if $\mathbb{S}$ is acquaintance-level serial w.r.t S_i.*

Proposition 1. *A global P2P schedule $\mathbb{S}$ consists of a set of global transactions $T = \{T_1, T_2, \cdots, T_n\}$ is serializable over a propagation path $(P_i \rightarrow, \cdots, \rightarrow P_z)$ with respect to a schedule S_i at P_i, if for each acquaintance in $(P_i \rightarrow, \cdots, \rightarrow P_z)$, $\mathbb{S}$ is acquaintance-level serializable.*

Proof. We can prove the proposition 1 by induction method considering each acquaintance between P_i to P_z.

Let l be the length of the propagation path of T from P_i to P_z.
Case $l = 0$: T executes only at P_i and there is no further propagation of T. According to our assumption that each local schedule is serializable. Hence the global P2P schedule $\mathbb{S}$, which consists of only the schedule S_i, is serializable.

Case $l = 1$: T executes at P_i and an acquaintee P_j of P_i over an acquaintance $(i \rightarrow j)$. Since $\mathbb{S}$ is serializable over a single acquaintance $(i \rightarrow j)$ according to Theorem 1, therefore serialization orders of T in S_i and S_j are same. Hence, $\mathbb{S}$ is serializable over the path $(P_i \rightarrow P_j)$.

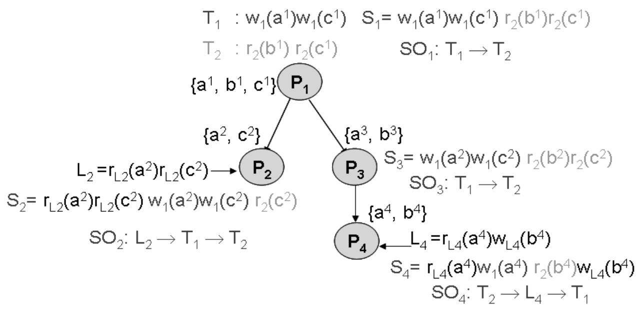


Fig. 6. Example of global P2P serializability

Case ($0 \leq k \leq l$): For the induction step we assume that serializability holds along the path between P_i and P_k recursively in each acquaintance, where P_k is a peer before P_z. Now we need to show that serializability holds between P_k and P_z, where $l = k + 1$. Since $\mathbb{S}$ is serializable over the path $(P_i \rightarrow, \cdots, \rightarrow P_k)$ and P_k and P_z are directly acquainted, therefore, $\mathbb{S}$ is serializable in $(P_k \rightarrow P_z)$. Hence, global serializability holds over the path $(P_i \rightarrow, \cdots, \rightarrow P_z)$.

Definition 8 (Global P2P Serializability). *A global P2P schedule $\mathbb{S}$ over a set of global transactions T initiated at P_i is globally serializable if*

1. *all local schedules in $\mathbb{S}$ are serializable and*
2. *for each acquaintance $(j \rightarrow k)$ over all the propagation paths $(P_i \rightarrow, \cdots, \rightarrow P_z)$, $\mathbb{S}$ is acquaintance-level serializable.*

Proposition 2. *A global P2P schedule $\mathbb{S}$ consists of a set of global transactions $T = \{T_1, T_2, \cdots, T_n\}$ is globally serializable with respect to an initial schedule S_i at P_i, if for all propagation paths $(P_i \rightarrow, \cdots, \rightarrow P_z)(1 \leq z \leq m)$ and for each acquaintance in each path $(P_i \rightarrow, \cdots, \rightarrow P_z)$, $\mathbb{S}$ is acquaintance-level serializable and each path between P_i and P_z is acyclic.*

Proof. According to proposition 1, $\mathbb{S}$ is serializable in a path of $T's$ propagation. That means T is serializable in each path $(P_i \rightarrow, \cdots, \rightarrow P_z)(1 \leq z \leq m)$. Also, each path is acyclic. Therefore, $\mathbb{S}$ is globally serializable.

Example 5. Consider the Figure 6 where two global transactions T_1 and T_2 are initiated at P_1. In the scenario, the global P2P schedule $\mathbb{S}$ is $\{\mathbb{S}_1, \mathbb{S}_2, \mathbb{S}_3, \mathbb{S}_4\}$. From the figure we notice that all the local schedules are locally serializable. The initial schedule S_1 has the serialization order $SO_1 = T_1 \rightarrow T_2$. The *acquaintance-level* schedule $\mathbb{S}_1$ with respect to S_1 at P_1 is $\{S_1, S_2, S_3\}$. We see that $\mathbb{S}_1$ is *acquaintance-level* serial with respect to S_1 because both SO_2 and SO_3 have the serialization order $T_1 \rightarrow T_2$. But $\mathbb{S}_3$ is not *acquaintance-level* serial because SO_4 at P_4 is not consistent with SO_3 of P_3. Therefore, $\mathbb{S}$ is not globally serializable.

6 Scheduling Transactions in a P2PDBS

Although there is no GTM in a P2PDBS, but the global serializability can be achieved by ensuring *acquaintance-level* serializability in each propagation paths of global transactions. In order to ensure *acquaintance-level* serializability, we need to guarantee the consistent serialization order of the transactions at all the acquaintees of a peer. We know that when a set of global transactions T is initiated at a peer P_i, the transactions are executed immediately at P_i. Therefore, P_i generates its local schedule S_i without waiting for the execution of T in its acquaintees. P_i then forward T to its acquaintees after translation. The execution and forward steps continue until no propagation of T is possible. Each of the peer P_j executes T with the local concurrency control and generates the local schedule S_j independently. The main challenge is how to guarantee the consistent serialization order in all S_j with respect to S_i. In the following, we propose two methods that ensures *acquaintance-level* serializability.

6.1 Merged Global Transactions (MGT)

When a set of global transactions T is initiated at a peer or is received by a peer then the peer immediately executes T with the local concurrency control protocol. Therefore, the peer generates a local serializable schedule forming a specific serialization order of the global transactions in T. The main goal of the P2P global serializability is that each participating peer must execute T in the same order as it is generated at the initiator. In this method, we assume that a function called $returnSchedule()$ is used by each peer that returns the locally generated schedule of T. Note that the local schedule may contain the operations of local transactions. The $returnSchedule()$ function returns only the operations of T in the same order appeared in the schedule. We assume that the function is added externally in the system. A peer treats the schedule returned by $returnSchedule()$ as an atomic transaction and translates the schedule for each of its acquaintee according to the data mappings. The peer then forwards the schedule as a new transaction to its acquaintees. When an acquaintee receives the schedule (now transaction), the acquaintee processes the schedule as it processes a transaction. Note that treating a schedule as a transaction keeps the order of operations of the original transactions since the order of the interleaved operations in the schedule remains same during translation according to Definition 2.

Now we describe the method with examples. We show that the *acquaintance-level* serializability is maintained considering both the direct and indirect conflict.

Example 6 (Direct Conflict). Consider the situation of Example 3. The $returnSchedule()$ function returns the following schedule generated at P_1.

$$S_1 = w_1(a^1)w_1(c^1)r_2(b^1)r_2(c^1).$$

According to the method, P_1 creates a transaction T_{12} for its acquaintees P_2 and P_3 from the schedule S_1. The order of operations of T_{12} follows the order as mentioned in the schedule S_1.

$$(P_2) : T_{12} = w_{12}(a^2)w_{12}(c^2)r_{12}(c^2),\ \ (P_3) : T_{12} = w_{12}(a^3)r_{12}(b^3)$$

Consider that P_2 and P_3 generate the following schedules when they receive T_{12}.

$$S_2 = r_{L2}(a^2)r_{L2}(c^2)w_{12}(a^2)w_{12}(c^2)r_{12}(c^2),\ S_3 = r_{L3}(a^3)w_{12}(a^3)r_{12}(b^3)w_{L3}(b^3)$$

Note that the schedule S_3 is not allowed by the local concurrency control of P_3. That is either L_3 or T_{12} will be blocked or aborted. On the other hand, if the local schedule at P_3 were

$$S_3 = r_{L3}(a^3)w_{L3}(b^3)w_{12}(a^3)r_{12}(b^3) \text{ or } S_3 = w_{12}(a^3)r_{12}(b^3)r_{L3}(a^3)w_{L3}(b^3)$$

then the schedule would be permitted by the local concurrency control at P_3 and therefore ensures serializability with respect to S_1. Note that the transaction T_{12} contains the operations of T_1 and T_2. As T_{12} is an atomic transaction for the transaction manager of P_2 and P_3, therefore the operations T_{12} are executed in the same order as executed at P_1. Hence, the serialization order of T_1 and T_2 must be same at P_1, P_2, and P_3. Therefore, $acquaintance - level$ serializability is maintained at P_1,P_2, and P_3 with respect to schedule S_1.

Example 7 (Indirect conflict). Consider the Example 4, there is no conflict between T_1 and T_2 when they were initiated at P_1. Assume that the following schedule is generated at P_1.

$$S_1 = w_1(a^1)w_2(b^1)w_2(c^1)$$

According to the method, P_1 creates the following transactions for its acquaintees P_2 and P_3 from the schedule S_1.

$$(P_2) : T_{12}{=}w_1(a^2)w_2(c^2),\ \ (P_3) : T_{12}{=}w_1(a^3)w_2(b^3)$$

Assume that the following schedules result at P_2 and P_3 respectively.

$$S_2 = w_{12}(a^2)r_{L2}(a^2)r_{L2}(c^2)w_{12}(c^2),\ S_3 = r_{L3}(a^3)w_{12}(a^3)w_{12}(b^3)r_{L3}(b^3)$$

Notice that the schedule S_2 and S_3 are not allowed by the local concurrency control of P_2 and P_3. In P_2, either L_2 or T_{12} will be blocked or aborted. If the local schedule in P_2 were

$$S_2 = r_{L2}(a^2)r_{L2}(c^2)w_{12}(a^2)w_{12}(c^2) \text{ or } S_2 = w_{12}(a^2)w_{12}(c^2)r_{L2}(a^2)r_{L2}(c^2)$$

then the schedule would be permitted by the local concurrency control at P_2.

Similarly, if the local schedule in P_3 were

$$S_3 = w_{12}(a^3)w_{12}(b^3)r_{L3}(a^3)r_{L3}(b^3) \text{ or } S_3 = r_{L3}(a^3)r_{L3}(b^3)w_{12}(a^3)w_{12}(b^3)$$

then the schedule would be permitted by the local concurrency control at P_3. Note that from the above execution, both P_2 and P_3 schedule the transactions

T_1 and T_2 logically in the same order. Therefore, the *acquaintance-level* serializability is maintained with respect to S_1.

6.2 Ticket Method

In this method we exploit the concept of ticket approach [10]. According to this protocol, a peer includes an extra operation $w(t)$ before the first operation of each transaction when the transactions are forwarded to the acquaintees. The $w(t)$ is a write ticket operation. A ticket is a (logical) timestamp whose value is stored as a regular data item in each LDBS [10]. The intuition behind the use of $w(t)$ operation is to create a relative serialization order of the global transactions. The inclusion of $w(t)$ operation does not violate the autonomy of LDBS nor does pose any restriction in the LDBS. The inclusion of $w(t)$ operation is outside the scope of local TM. We also assume that there is a function called $getSeriliazeOrder()$ that returns the serialization order of the executed global transactions in the local peer. When the peer forwards the global transactions, it sends them according to the serialization order as returned by the function $getSeriliazeOrder()$. This can be performed simply by delaying the transactions. When a remote peer receives the transactions it processes them accordingly and is allowed to interleave the operations of the transactions under the control of the LDBS. Note that, adding the $w(t)$ operation creates a direct conflict between the transactions at remote peers. Therefore, transactions will be serialized in the remote peer as determined by the sender.

In the following we describe the protocol using an example considering that there is a no conflict between transactions when they are initiated at a peer.

Example 8 (Indirect conflict). Consider the Example 4. The $returnSchedule()$ function returns the following schedule at P_1.

$$S_1 = w_1(a^1)w_2(b^1)w_2(c^1).$$

According to the method, P_1 creates the following transactions adding $w(t)$ operation to T_1 and T_2 before forwarding them to the acquaintees P_2 and P_3.

$$(P_2){:}T_1{=}w_1(t)w_1(a^2),\ T_2{=}w_2(t)w_2(c^2),\ (P_3){:}T_1{=}w_1(t)w_1(a^3),\ T_2{=}w_2(t)w_2(b^3)$$

P_1 uses the $getSeriliazeOrder()$ function to find the serialization order of T_1 and T_2. From the schedule we see that T_1 is executed before T_2 in S_1. Therefore, P_1 sends T_1 before T_2 to its acquaintees P_2 and P_3 respectively.

Consider that the following schedule are generated by P_2 and P_3 after receiving T_1 and T_2.

$$S_2 = w_1(t)w_1(a^2)r_{L2}(a^2)r_{L2}(c^2)w_2(t)w_2(c^2),$$
$$S_3 = r_{L3}(a^3)w_1(t)w_1(a^3)w_2(t)w_2(b^3)w_{L3}(b^3)$$

Note that the schedule S_3 is not allowed by the local concurrency control of P_3. That is either L_3 or T_2 will be blocked or aborted. On the other hand, if the local schedule in P_3 were

$$S_3 = w_1(t)w_1(a^3)w_2(t)w_2(b^3)r_{L3}(a^3)w_{L3}(b^3),$$
$$S_3 = w_1(t)w_1(a^3)r_{L3}(a^3)w_{L3}(b^3)w_2(t)w_2(b^3)$$

then the schedule would be permitted by the local concurrency control at P_3 and therefore ensures *acquaintance-level* serializability. Similarly, for example 3, we can show that *acquaintance-level* serializability can be maintained using the ticket method.

7 Implementation Issues

We are developing the transaction processing framework to incorporate into the Hyperion [2] P2PDBS system as a transaction service. The underlying P2P network is created using the JXTA framework and each peer contains a MySQL database. We consider a transaction as a transaction service that contains either a single or a group of SQL commands (e.g. Select, Update, etc.). In the following we describe the different components of the service module:

– **Transaction Service:** A transaction service operates in two modes: local or remote. In the local mode, the service processes request locally without requiring assistance of other peers; in the remote mode, the service needs remote resources (i.e. acquaintance to other peers and services offered by those peers [transaction service]) to finish its job.
– **Transaction Service Manager:** Each peer has a transaction service manager that handles execution of transactions. The transaction service manager takes care of transactions that are received from the local as well as from remote peers. Transaction service manager creates a transaction handler for each of the transactions.
– **Service Handler:** A Transaction Service Manager can coordinate multiple transactions simultaneously by using transaction Service Handlers. When a Transaction Service Manager receives a transaction, it first creates a Service Handler for that transaction and passes over the transaction to the Handler. Afterwards, the Handler uses the LDBS to process the transaction; the service continues to wait for a new transaction.

7.1 Typical Transaction Service Request Scenario

1. User on peer A submits a transaction execution request. The Transaction Service Manager creates a service handler to process the request.
2. The service handler processes the request locally and asks the transaction service manager to find the remote resource for execution over peer A's acquaintances.
3. The transaction service manager invokes acquaintance service to get acquaintances of peer A.
4. The manager then consults with Transaction Translation Component to translate the transaction for all the acquaintances and sends the translated transactions to the corresponding acquaintees.

5. The acquaintee peer B receives the request and forwards to the Transaction Service Manager. The Service Manager of peer B then creates a transaction handler to process the transaction locally.
6. The Service Handler of peer B then sends a response to the Transaction Service Manager. The Transaction Service Manager of peer B then sends the response to peer A.
7. The Transaction Service Manager wakes up the waiting Handler and delivers the response message to it.
8. The Handler finishes processing the transaction and gives the result back to the Transaction Service Manager.
9. Transaction Service Manager notifies the waiting user that the transaction is processed.

8 Related Work

Until now there have been many algorithms proposed for transaction management for MDBSs. The solutions are not directly applicable in P2PDBSs due to the absence of a global coordinator in the systems and arbitrary topology of P2P networks. Recently, very few researchers attempted to focus on transaction processing in P2P networks. Now we describe some of the solutions.

In [13,12] a concept of transaction processing in P2P environment is presented. The authors proposed a decentralized concurrency control for transactions relying on a decentralized serialization graph. Each peer and each transaction maintain a local serialization graph. The serialization graph of the peer reflects the dependencies of the transactions that invoked service calls on that peer whereas the serialization graph of the transaction includes the dependencies in which the transaction is involved. However, the strategy needs to modify the underlying database system to support the protocol. Also, it needs an application layer for creating the transactions agents, managing lock table, and processing the serialization graphs. In this paper, we assume that underlying system remains unchanged. In [15], authors present a preliminary proposal for peer-to-peer e-business transaction processing system. More specifically, the paper focuses on requirements analysis on different aspects of the collaboration and transaction procedure. However, it lacks precise semantics of transactions and does not describe the execution semantics of transactions. In [14] a preliminary approach for agent-based transaction in a decentralized P2P network is presented. The focus is on a cooperative information system based on a P2P model. The model consists of a multi agent system with four components(wrapper, mediator, facilitator, and planner) which are responsible for the management and control of transactions composed by data management operations (read, write, delete) and their outcomes. However, the approach has no details.

9 Concluding Remarks

In this paper, we introduced a transaction model for a P2PDBS where sources are heterogeneous and instance level mappings are used to associate data from different sources. Our approach is scalable because a peer doesn't need any global knowledge of the system and there is no global coordinator. Transactions are processed by each peer independently and consistency is maintained recursively through acquaintances. A peer only ensures the serializability of its immediate acquaintances by ensuring $acquaintance - level$ serializability. Mainly, we contribute the following:

• analyze the properties and semantics of transactions in a P2PDBS.

• show a correctness criterion that ensures the consistency of a P2PDBS during concurrent execution of transactions initiated from a single peer.

• propose two approaches ensuring global serializability without violating the autonomy of LDBSs and describe the implementation scenario to demonstrate the approaches.

A future goal is to investigate the transaction processing when global transactions initiated form many peers need to be executed concurrently in the system and analyze the correctness criterion for such executions. Finally, we want to investigate these problems in a large peer network and show the scalability of the system.

References

1. Kementsietsidis, A., Arenas, M., Miller, R.J.: Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In: SIGMOD (2003)
2. Hyperion Project. World Wide Web, http://www.cs.toronto.edu/db/hyperion/
3. Halevy, A.Y., Ives, Z.G., Madhavan, J., Mork, P., Suciu, D., Tatarinov, I.: The piazza peer-data management system. IEEE Transactions on Knowledge and Data Engineering 16(7) (2004)
4. Alonso, R., Garcia-Molina, H., Salem, K.: Concurrency Control and Recovery for Global Procedures in Federated Database Systems. IEEE Data Engineering Bulletin 10(3) (1987)
5. Breitbart, Y., Garcia-Molina, H., Silberschatz, A.: Overview of multidatabase transaction management. VLDB Journal 1(2) (1992)
6. Breitbart, Y., Silberschatz, A., Thompson, G.R.: Transaction Management Issues in a Failure-Prone Multidatabase System Environment. VLDB Journal 1(1) (1992)
7. Du, W., Elmagarmid, A.: Quasi serializability: a correctness criterion for global concurrency control in InterBase. In: VLDB (1989)
8. Wolski, A., Veijalainen, J.: 2PC Agent Method: Achieving Serializability In Presence Of Failures In A Heterogeneous Multidatabase. In: PARBASE (1990)
9. Mehrotra, S., Rastogi, R., Breitbart, Y., Korth, H.F., Silberschatz, A.: Overcoming Heterogeneity and Autonomy in Multidatabase Systems. Information and Computation 167(2) (2001)
10. Georgakopoulos, D., Rusinkiewicz, M., Sheth, A.: Using Tickets to Enforce the Serializability of Multidatabase Transactions. IEEE Transactions on Knowledge and Data Engineering 6(1) (1994)

11. Breitbart, Y., Silberschatz, A.: Multidatabase update issues. In: ACM SIGMOD, ACM Press, New York (1988)
12. Haller, K., Schuldt, H., Türker, C.: Decentralized Coordination of Transactional Processes in Peer-to-Peer Environments. In: CIKM (2005)
13. Türker, C., Haller, K., Schuler, C., Schek, H.: How can we support Grid Transactions? Towards Peer-to-Peer Transaction Processing. In: CIDR (2005)
14. Penserini, L., Panti, M., Spalazzi, L.: Agent-Based Transactions into Decentralised P2P. In: AAMAS (2002)
15. Androutsellis-Theotokis, S., Spinellis, D., Karakoidas, V.: Performing peer-to-peer e-business transactions: A requirements analysis and preliminary design proposal. In: IADIS (2004)

Enabling Selective Flooding to Reduce P2P Traffic

Francesco Buccafurri and Gianluca Lax

DIMET, Università degli Studi Mediterranea di Reggio Calabria
Via Graziella, Località Feo di Vito, 89122 Reggio Calabria, Italy
bucca@unirc.it, lax@unirc.it

Abstract. We propose a P2P cooperation policy to increase the effectiveness of the flooding-based approach used to retrieve information over pure P2P networks. Flooding consists in propagating the original query from the source peer to "known" peers, and, in turn, to other peers, producing, in the general case, an exponential grow of the search traffic in the network. According to our policy, each peer involved in the flooding process propagates the query only toward peers hopefully capable of satisfying it. The crucial point is: how to detect such good candidates? Of course, "local" properties of similarity between peers not satisfying the transitivity property cannot be used to the above purpose, due to the necessity of propagating queries. Our solution relies on recovering some transitivity behavior in similarity-based P2P information retrieval approaches by considering neighborhood semantic properties. Experimental results show that the selective flooding so obtained is effective in the sense the traffic is drastically reduced w.r.t. the standard flooding (like GNUTELLA), with no loss of query success.

1 Introduction

Peer-to-Peer is a class of systems using distributed resources in a decentralized and autonomous manner. They are exploited in many contexts such as market and demographic analysis, code breaking, risk hedge calculations, genome sequence, protein folding, instant messaging, file mirroring, online storage, file sharing, and many others. Whenever objects are searched in the P2P system with no unique key (i.e., in case of *partial queries*), the common approach, called *flooding*, is submitting the query to the neighborhood of the source peer, and forwarding it until a stop condition occurs. More precisely, a peer P, looking for an item X, broadcasts a message containing a request for X to all its neighbors. In turn, each peer receiving the request, iterates the above protocol, by possibly answering to P and by forwarding the message to its neighbors. For example, this is essentially what GNUTELLA [14] does. In order to guarantee finiteness of the protocol, both the life time and the number of times the query is forwarded (*hops*) as well as the list of the crossed peers, may be stored into the query, and used for halting its propagation. Unfortunately, the above approach produces an exponential grow of messages generated by a query in the number of hops. Thus, propagation of a query must be significantly limited, by setting a small maximum number of hops (typically up to 10) and/or a little query life time. Clearly, the more propagation is limited the lower expectation about query success is. As a consequence, it is a very interesting issue to investigate methods for reducing the number of peers the query is forwarded to (at each hop).

In this paper we propose a P2P cooperation policy, whose purpose is increasing the effectiveness of approaches commonly used for retrieving information over pure P2P networks. We refer to the case of partial queries in absence of directory services (i.e., in pure P2P systems) [1]. Our proposal is based on the detection, for each peer involved in the query propagation, of the k-most promising peers which the query has to be forwarded to. Both the value of k and, more importantly, the way of determining such k-most promising peers belong to a common policy, voluntarily adopted by the community. We consider the case of non-critical activities performed in the P2P system. Moreover, as it will be clear in the following, both condition (a) and condition (b) stated above are verified. In particular, condition (a) is guaranteed by the improvement of the effectiveness of information search, making the approach advantageous for every member (this is shown in the paper by simulation), and condition (b) is satisfied since the failure of some member to comply with the policy (by enabling a standard non-filtered flooding) produces only negative effects locally restricted to the member itself.

The basic problem is of course which strategy the policy has to rely on, in order to reduce the traffic generated by the flooding process and, consequently, improving the information search. Interesting approaches are those the above "blind" search is substituted by a semantic-driven one. However, due to query propagation, a semantic-driven approach based on similarity between peers may be effective only if such a property presents some form of transitive behavior. Indeed, only in this case, the peer pruning strategy can be applied at each stage of the query propagation. On the other hand, semantic embedded into the query can be used in order to propagate information useful for peer pruning (see [7] as an example of approaches of this type). However, in order that the query can carry a sufficient semantic power, its structure, to be compared with the peer content, should be enough rich and complex.

We consider the real-life case of simple retrieval queries, so we do not want to assume that the information embedded into the query can be exploited for peer pruning. Beside applicability to general real-life contexts, a similar approach has the important advantage that peer selection can be performed off-line, before the query is submitted. Even though we fall into the case of peer similarity-based approaches, we propose a model for extracting, instead of just local similarities (i.e., similarities between peers), *neighborhood* semantic properties, embedding similarities, and presenting some form of transitivity in their behavior. The term neighborhood means that what we use is not simply the similarity of a peer w.r.t. another peer, but also the closeness of the latter w.r.t. the community of

[1] [13] is an example of P2P system in which no node is more important than any other node, while, for instance [19] uses a central directory service.

peer enough similar to the former. In addition, the neighborhood-based approach shows nice properties of adaptivity to user interests, without producing unstable behaviors.

The paper is organized as follows. Section 2 illustrates how resources of peers are described and how similarity between two peers is defined. In Section 3 we describe the core of our proposal: We introduce the notion of expectation and we define the mathematical framework allowing us to represent semantic closeness among peers. In Section 4 we describe how adaptivity is implemented in our system, while Section 5 illustrates through an extensive example how the method works. In Section 6 we provide a number of experiments we have conducted in order to validate the proposed method. In Section 7 we survey the main proposal related to our work and finally, in Section 8 we draw our conclusions.

2 Peer Similarity

Peer similarity is based on resources contained in the peers. Our approach does not rely on sophisticated ontology models shared by the peers, but on a simple model in which resources (for example music files) are represented by a number of metadata (like for instance author, category , etc.) along with their value (e.g., Pink Floyd, Rock, etc.) and their occurrence (e.g., 30, 850, etc., meaning that such a peer contains 30 Pink Floyd's songs, 850 files belonging to the category Rock, etc.). We describe more precisely this model next. Now we observe that the choice of this simple model arises from two considerations. First, the main aspect of our proposal regards the use of the expectation property (defined in the following) allowing us to overcome limits of similarity-based systems. Therefore, we have chosen to introduce the expectation in a context as simple as possible. The second advantage arising from our choice is making our approach applicable to real-life P2P commercial environments where resources are basically described in the same way as our model. Anyway, our approach is parametric w.r.t. the used ontology model and, consequently, other models, possibly more sophisticated, like [12,4,7,25], could be exploited without changes (beside, obviously, the computation of the similarity).

Our ontology model is defined by a finite set $\mathcal{U} = \{M_1, \ldots, M_m\}$ of possible metadata. Each metadata M_l can assume a value within its domain $D(M_l)$. The content of each peer is described according to this shared ontology model. Each resource is associated with a non empty subset of $\mathcal{U}$ (i.e., a number of metadata).

Given a peer P, for each pair (M_l, V), where M_l is a metadata occurring in (some resource of) P and $V \in D(M_l)$, we denote by $Num_P(M_l, V)$ the number of resources of P with metadata M_l assuming the value V. The pair (M_l, V) is said *assignment for* M_l. For example, if a peer P contains 23 files having the value *classical* for the metadata *type*, then $Num_P(\textit{type, classical}) = 23$.

Given a positive integer s, the *(s-)content descriptor* of a peer P is the set of pairs $\langle (M_l, V), Num_P(M_l, V) \rangle$ such that M_l is a metadata occurring in P and $Num_P(M_l, V) \geq s$. s represents a threshold, making explicit the minimum number of occurrences of metadata instances, thus allowing us to represent only metadata sufficiently relevant. The default used in this paper for s is 1.

Example 1. An example of 5-content descriptor of a peer whose user is interested in classical music might be the following: $(\langle type, classical, 23\rangle, \langle author, Mozart, 10\rangle, \langle author, Bach, 7\rangle)$. Observe that since $23 > 10 + 7$, in this peer there are also other classical music authors, but the number of occurring pieces of each one is not considered relevant (i.e., it is less than 5).

The problem we have to consider now is defining a suitable notion of similarity between two peers. When similarity notion has to be designed, a starting question should have to be considered. Has this relation S to be a *fuzzy similarity* relation [28]? That is, has it to be reflexive (i.e., $S_{ii} = 1$) , symmetric (i.e., $S_{ij} = S_{ji}$) and transitive (i.e., $S_{ij} \geq S_{ik} \oplus S_{kj}$, where $\oplus$ is any *T-norm* like the minimum function)? While renouncing to the first two properties would be unfounded, the last property is often missed (under plausible T-norms as the minimum), every time similarity is computed on the basis of possibly independent dimensions used for representing the content of the peers. In words, it is acceptable that we want to capture the case: A is very similar to B on the basis of some dimensions belonging to their description, B is very similar to C, but A is not similar to C, since dimensions which the similarity between B and C is based on are different from those supporting the closeness between A and B.

Let introduce now the notion of peer similarity. Let P_i and P_j be two peers with content descriptors O_i and O_j, respectively. The similarity $S_{i,j}$ between P_i and P_j is defined as: $S_{ij} = \frac{\sum_{(M_l,V)\in O_i \cap O_j} min(Num_{P_i}(M_l,V), Num_{P_j}(M_l,V))}{min(\sum_{(M_l,V)\in O_i} Num_{P_i}(M_l,V), \sum_{(M_l,V)\in O_j} Num_{P_j}(M_l,V))}$.

The numerator computes the number of resources of the two peers having the same value for the same metadata. The denominator normalizes the result so that similarity value lies into the range $[0, 1]$. In the following example we show how the similarity is computed and we show that our notion of similarity (but, we guess, every reasonable similarity for our ontology model) is not transitive (under the T-norm minimum).

file/peer	*wine*	*spir.*	*food*	*nutr. sc.*
P_1	60	42	6	0
P_2	48	0	65	16
P_3	13	0	64	34

Fig. 1. File distribution

simil.	P_1	P_2	P_3
P_1	1	0.50	0.18
P_2	0.50	1	0.84
P_3	0.18	0.84	1

Fig. 2. Peer similarity

Example 2. Consider three peers P_1, P_2 and P_3, having for the metadata category the following values: *wine*, *spirits*, *food* and *nutritional science*, as reported in Figure 1. Similarity coefficients between any pair of peers is reported in Figure 2. For example, the similarity between P_1 and P_3 is obtained by: $S_{13} = \frac{13+6}{min(108,111)} = \frac{19}{108} = 0.18$. As it can be verified by looking at Figure 3, this is an example of non transitivity (under T-norm minimum) of our notion

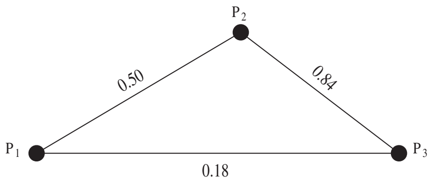


Fig. 3. Graph description

of similarity. Indeed, $S_{13} < minimum(S_{12}, S_{23})$. Intuitively, it happens that the peer P_1 is significantly similar to the peer P_2 thanks to the category *Wine*, while the high similarity between P_2 and P_3 relies on the community of interests in *Food*. Thus, there is no intersection of interests about P_1 and P_3, leading to a low similarity between such a pair of peers.

But, who does compute the similarity among peers? Each peer has to compute the similarity between itself and its neighbors. For this purpose, peers exchange their content-descriptor each other. In particular, in order to reduce the traffic overhead, each peer stores two different content-descriptors. The first one, called CCD (current content descriptor), is continuously updated accordingly to the new resources shared. The second one, called LSCD (last sent content descriptor), is initially equal to CCD. Each peer sends to its neighbors its CCD every time the similarity between CCD and LSCD is less than a fixed threshold δ. Then LSCD is update to CCD.

3 Beyond the Similarity: The Expectation Notion

The basic purpose of our approach is reducing the number of peers which the query is propagated to at each hop. As explained, this can be done by exploiting semantic properties. However, it is not possible to use the similarity *tout-court* in order to reach the above goal. Indeed, after the submission of the query to adjacent enough similar nodes, similarity cannot be exploited anymore, since it is not transitive. More precisely, if a peer B is chosen by a peer A according to its similarity, a generic peer C which the query has to be forwarded to, of course cannot be chosen on the basis of the similarity w.r.t. B, since we cannot argue anything about the similarity between A and C.

The aim of this section is (recursively) defining a new property, called *expectation* of a given peer A w.r.t. another peer B, by taking into account not only similarity between A and B, but also the expectations of all other peers C, enough similar to B. Expectation, as well as similarity, is represented as a fuzzy coefficient (i.e., it lies into the range $[0, 1]$). In words, expectation, from an *anthropomorphic* point of view of a peer A, captures something like: *if all the peers C, which my expectation w.r.t. is high, are highly similar to a peer B, then my expectation w.r.t. B has to increase, possibly compensating a low similarity of B w.r.t. me.*

By using expectation, we obtain the double advantage of: (1) giving to the method stability properties, due to the insertion of a network of logical connection from a peer to another peer instead of a single connection (this issue will be analyzed by simulation, in Section 6), and (2) giving to such a semantic property a sort of transitive behavior, overcoming the limit of the pure similarity (this issue will be better explained by an example in the following).

The informal notion described above, is defined through a linear system as follows. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ a set of peers. Let P_i, with $i > n$, be a peer not occurring in $\mathcal{P}$. Consider given $\mathcal{P}$ and P_i for the rest of the paper. The tuple $\langle E_{i1}, E_{i2}, \ldots, E_{in}\rangle$ of the expectations of P_i w.r.t. the peers in $\mathcal{P}$, is the unique solution of the following linear system:

$$(\forall j \in \{1, \ldots, n\}) \left(E_{ij} = \alpha_{ij}(\beta_i S_{ij} + (1 - \beta_i) \tfrac{1}{\hat{S}_j} \textstyle\sum_{k=1\ldots n, k \neq j} E_{ik} S_{kj}) \right) \qquad (1)$$

where (1) $\hat{S}_j = \sum_{k=1\ldots n, k\neq j} S_{kj}$ and (2) $0 \leq \alpha_{ij} \leq 1$ and $0 \leq \beta_i \leq 1$ are suitable coefficients initialized to 1.

Before giving the intuitive explanation of the above system, it is worth noting that the above definition is well founded, since the system admits always a unique admissible solution (that is, consisting of values ranging from 0 to 1). This is in fact stated next in Theorem 1.

Let give now an informal description of how the system works. Initially, do not consider coefficients α_{ij} and β_i, which will be explained later. E_{ij} is obtained by summing two contributions. The first one (i.e., S_{ij}) is the similarity between P_i and P_j. It can be considered a *local* component. The second one (i.e., $\frac{1}{\hat{S}_j} \sum_{k=1\ldots n, k\neq j} E_{ik} S_{kj})$)), makes E_{ij} neighborhood-dependent, that is dependent on the whole set $\mathcal{P}$ of peers. Thus, this component is said *neighborhood-based*. It increases E_{ij} by a contribution proportional to the expectancy of P_i w.r.t. any peer P_k, for each k (different from j). In order to make relevant only contributions relating to peers enough similar to P_j, the more the similarity between P_k and P_j, the higher the coefficient weighting such a contribution (i.e., $\frac{S_{kj}}{\hat{S}_j}$) is. In this way the overall expectation E_{ij} of P_i w.r.t. P_j takes into account not only similarity between P_i and P_j, but also the expectations of all other peers P_k, "enough" similar to P_j. According to this mechanism, the linear system can be viewed as a sort of set of *equilibrium* equations for the whole system.

Which is the role of α_{ij} and β_i? α_{ij}, as it will appear more clear in the next section, produces the reduction of expectation w.r.t. peers failing queries. It represents an *aging* coefficient. β_i modulates the importance of the local component w.r.t. the neighborhood-based one, and, as it will be explained in the next section, depends on the dynamics of the P2P system.

We are ready to present the theorem supporting the definition of expectation. This theorem shows that, for every value of the parameters occurring in (1), there exists a unique solution of the linear system (1), that is a value assignment to the expectation coefficients satisfying (1). Obviously, such a solution can be polynomially computed.

Theorem 1. *Given a set of real coefficients* $\{\alpha_{ij}$*,* β_i*,* S_{ij}*,* S_{jk} *such that* $\alpha_{ij} \in (0,1)$*,* $\beta_i \in (0,1)$*,* $S_{ij} \in [0,1]$*,* $S_{jk} \in [0,1]$*, with* $j \in \{1,\dots,n\}$ *e* $k \in \{1,\dots,n\} \setminus \{j\}\}$*, there exists a unique n-tuple of* $[0,1]$ *real values* $S = \langle E_{i1},\dots,E_{in}\rangle$ *satisfying (1).*

Proof. We first prove that there exists unique a solution of the system (1). Then, we show that such a solution is a tuple of fuzzy values (i.e., ranging from 0 to 1).

Existence and Uniqueness. It suffices to prove that the coefficient matrix M of the system (1) has maximum rank (i.e., its determinant is null). Denote by $m_{hk} = \alpha_{ih}(1-\beta_i)\frac{S_{kh}}{\hat{S}_j}$ a generic element of M, where $0 \le \sum_{r=1\dots n, r\ne h} m_{hr} < 1$. Then, M can be rewritten in the following way:

$$M = \begin{bmatrix} -1 & m_{12} & \dots & m_{1n} \\ m_{21} & -1 & \dots & m_{2n} \\ \dots & \dots & \dots & \dots \\ m_{n1} & m_{n2} & \dots & -1 \end{bmatrix}$$

By contradiction suppose the determinant of M is null. Thus, there exists a linear combination of the columns of M with non null coefficients $\langle h_1,\dots,h_n\rangle$, producing the 0-tuple. In particular, by considering the $k-$th row, with k such that h_k has the maximum value among $\langle h_1,\dots,h_n\rangle$, we obtain:

$$-h_k + \sum_{r=1\dots n, r\ne k} m_{ir}h_r = 0 \quad \Rightarrow \quad h_k = \sum_{r=1\dots n, r\ne k} m_{ir}h_r$$

Let s be such that h_s assumes the 2^{nd} maximum value among $\langle h_1,\dots,h_n\rangle$. Then, it follows that: $h_k \le h_s \sum_{r=1\dots n, r\ne s} m_{ir}$.

Applying the summation for $\sum_{r=1\dots n, r\ne h} m_{hr} < 1$, it results $h_k < h_s$, that is a contradiction. This concludes this part of the proof. Now it remains to prove that the solution is a tuple of values ranging from 0 to 1.

First we show that the solution is a tuple of values greater or equal than 0.

$\mathbf{E_{ih} \ge 0 \ \forall h \in \{1,\dots,n\}}$**.** By contradiction suppose there exists in the solution of (1) a negative expectation. Thus, if E_{im} denotes the minimum expectation of the solution, then $E_{im} < 0$. Therefore, by (1) we obtain:

$$E_{im} = \alpha_{im}\left(\beta_i S_{im} + (1-\beta_i)\frac{1}{\hat{S}_m}\sum_{k=1\dots n, k\ne m} E_{ik}S_{km}\right) < 0$$

Since E_{im} is the minimum expectation and α_{im} ranges from 0 to 1, we have that:

$$E_{im} \ge \alpha_{im}\left(\beta_i S_{im} + (1-\beta_i)\frac{1}{\hat{S}_m}\sum_{k=1\dots n, k\ne m} E_{im}S_{km}\right) \ge \beta_i S_{im} + (1-\beta_i)E_{im}$$

and thus: $\beta_i E_{im} \ge \beta_i S_{im} \quad \Rightarrow \quad E_{im} \ge 0$.
We have thus reached a contradiction, concluding this part of the proof.

The last step is showing that the solution is a tuple of value less or equal than 1.

$\mathbf{E_{ih} \leq 1\ \forall h \in \{1, \ldots, n\}}$. By contradiction suppose that there exist in the solution at least one expectation greater than 1. Denote by E_{iM} be the expectation having the maximum value. Necessarily, $E_{iM} > 1$. Since we have already proven that $E_{ih} \geq 0\ \forall h \in \{1, \ldots, n\}$, from (1) we obtain:

$$\alpha_{iM}(\beta_i S_{iM} + (1-\beta_i)\frac{1}{\hat{S}_M} \sum_{k=1\ldots n, k\neq M} E_{ik} S_{kM}) \leq \beta_i + (1-\beta_i)\frac{1}{\hat{S}_M} \sum_{k=1\ldots n, k\neq M} E_{iM}$$

and, thus: $E_{iM} \leq \beta_i + (1-\beta_i)E_{iM} \quad \Rightarrow \quad E_{iM} \leq 1$, that is a contradiction. The theorem is then proven. ■

In the next example we show that the above mechanism gives to the expectation based relation used for measuring the "semantic closeness" between two peers, a sort of transitive behavior, which is not satisfied whenever just the similarity is adopted (as shown in Example 2).

Example 3. Consider the set of peers introduced in Example 2. Suppose that $\beta_1 = 0.22$, $\alpha_{12} = 0.87$, $\alpha_{13} = 0.92$, meaning that the attention of P_1 toward the expectation parameter is considerable (and thus, the importance the peer gives to the similarity is low), and furthermore, both P_2 and P_3 have satisfied queries submitted by P_1 in the recent past (since the α coefficients are high). The last fact denotes that even though P_1 and P_3 are structurally not similar (as the reader may found in Figure 1, P_1 is mainly concerned with alcoholic beverage, while P_3 is interested in information – even scientific – about food), there could be an interest of P_1 in the resources shared by P_3 about, for example, both information about food and information about nutrition science, since the user of P_1 is now interested in right combination between food and wine as well as in nutritional information about wine. The solution of the linear system produces in this case the following result: $\langle E_{12} = 56.25\%, E_{13} = 43.75\%\rangle$, indicating that the peer P_3, despite its low similarity, is appealing for P_1 nearly as much as the very similar peer P_2. Observe that, by considering similarities, being $S_{12} = 73.53\%, S_{13} = 26.47\%$, the attractiveness of the peer P_3 w.r.t. P_1 would have been dramatically smaller than P_2 one (in fact, this reflect the non-transitive behavior of the similarity, shown in Example 2).

The example above shows that, whenever the dynamics of the system gives us knowledge contrasting with the quasi-static information embedded in the content descriptor, the expectation-based approach allows us to exploit this knowledge, by emphasizing semantic closeness not discovered by the similarity-based analysis. In other words, even though we accept the non transitivity of the similarity notion (since, as already discussed, we require the capability of encoding a closeness measure based on possibly orthogonal dimensions), there are cases in which the query history shows that such a transitivity has to be in a certain measure recovered, so that we have to use something more than just the similarity.

We remark that the above linear system does not capture the whole P2P system, but just the view a peer P_i has of it. Indeed, the set $\mathcal{P}$ represents the set of adjacent peers of P_i (i.e., those peers whose IP is known to P_i). As a consequence, every peer occurring in the system has a different view of it, corresponding with a different set of adjacent peers and, thus, a different linear system. It is worth noting that, as will be more clear in Section 5, the set of adjacent peers constantly tends to increase, since every time a peer P_j responds to a query coming from a peer P_i, P_j is inserted (if not already present) into the set of adjacent peers of P_i. As usual in P2P systems, in order to avoid an inflationary grow of its dimension, the set of adjacent peers must be pruned, by fixing a maximum allowed size. To this aim, standard approaches can be directly applied to our case, but of course, the low expectation can be used as a criterion for replacing peers when needed. Another parameter that our policy has to manage, for taking into account QoS features, is the maximum allowed set of contacts (i.e., peers, among adjacent ones, which queries are forwarded to). Throughout this paper we often call this parameter *selection size*. Thus, denoting by k the selection size, every peer P_i complying to the policy, first computes the expectation w.r.t. its adjacent peers and then, among these, forwards queries only to the $k-$first peers according to the expectation value.

Concerning the computational complexity, we observe that the only relevant task to analyze is the solution of a linear system like (1). This is a classic well studied problem, and many algorithms have been proposed for making feasible its solution also for large system dimensions. These results are of high practical interest, since large systems of linear equations occur in many applications such as finite element analysis, power system analysis, circuit simulation for VLSI CAD, and so on. In the general case, the cost for finding an exact solution is $\mathcal{O}(n^\omega)$, coinciding with the cost of executing a $n \times n$-matrix product. The currently best known ω is 2.376 [8], while a practical bound is 2.81 [23,6]. We observe that the coefficient matrix of our linear system is sparse in many practical situations. Indeed, it arises from the set of contacts of peers in the system forming typically weakly connected components. Note that for sparse linear systems, more efficient solutions may be found. In [21] it is shown that the cost is $\mathcal{O}(n + s(n)^\omega)$, where $s(n)$ is a function measuring the sparsity of the system's coefficient matrix (note that $s(n) = \mathcal{O}(n)$, in the general case). Beside exact (*direct*) methods considered above, there exists a wide variety of *iterative* methods, whose effectiveness strongly depends on the matrix sparsity. In [21] an evaluation of the upper bound of the number of iterations needed for the convergence of an $\epsilon-$solution (for relative error ϵ) is provided. Each iteration has in general the cost $\mathcal{O}(n^\omega)$ since it requires a matrix product. Multigrid methods can be applied [1,20] in order to decrease the cost per iteration until $\mathcal{O}(n^\omega)$, even though such methods are not applicable in general and may suffer of instability problems.

In Section 6 we have performed experiments in order to test the efficiency of the computation of the expectations, obtaining the result that such a computation is feasible also in case whose dimension is significant in real-life contexts.

4 System Adaptivity

Adaptivity of the system is implemented by the updating rules of the coefficients α_{ij} and β_i appearing in the linear system. What we want the system learns, while it works, is both (1) the success degree in finding query results in high-expectation peers and (2) the influence of local similarity w.r.t. the success degree. Discovering (1) allows us to progressively reduce (by reducing α_{ij}) the expectation of a peer w.r.t. another peer, if no positive answer comes from this peer anymore. The purpose of (2) is adapting (by modifying β_i) the importance of the similarity w.r.t. the neighborhood-based component.

W.l.o.g, think to the P2P system seen by P_i like a discrete-time system such that at the generic intermediate step k the following events occur in order: (1) the answers to the query submitted by P_i at the previous step are received by P_i, (2) accordingly, α and β parameters are updated, (3) a new query is submitted by P_i. Initially $\alpha_{ij} = 1$ for each $1 \leq j \leq n$. We expect that every time the peer P_j is not able to answer to the query of P_i, the coefficient α_{ij} is decreased. However, according to a *locality principle*, the reduction of α_{ij} should be limited in case P_j is enough similar to some peer that, in the last step, has satisfied the query of P_i. Conversely, the decrease of α_{ij} should be bigger, in the other case. Certainly, if P_j satisfies the query of P_i, α_{ij} is reset to 1. The above mechanism can be captured by the following updating rule, where the dependency on the time instant is made explicit. Thus, at the instant time k, if P_j has not satisfied the query submitted by P_i at the instant time $k-1$, then $\alpha_{ij}(k)$ is obtained by: $\alpha_{ij}(k) = \frac{\alpha_{ij}(k-1)+\alpha_{ij}(k-1)\cdot max_{P_h \in Y_{ik}}(S_{hj})}{2}$, where Y_{ik} denotes the set of peers which have satisfied the query of P_i submitted at the time instant $k-1$. Otherwise (i.e., if $P_j \in Y_{ik}$ – that is, it has responded to the query at the previous step) $\alpha_{ij}(k) = 1$. Concerning the coefficient β_i, initially set to 1, the value at the time instant k has to be increased if, at the preceding instant, success peers was similar to P_i. β_i has to be decreased, otherwise. Also this choice satisfies a sort of locality principle w.r.t. user interests. Thanks to this mechanism we avoid that the system continues to force P_i towards similar peers (thus towards interests consolidated into its content) in case the recent query history shows that peers useful for P_i are other peers, so that interests of P_i are changing. The adopted updating rule is: $\beta_i(k) = \frac{\beta_i(k-1)+\frac{1}{|Y_{ik}|}\sum_{P_h \in Y_{ik}}(S_{hi})}{2}$.

Observe that both for α_{ij} and β_i, values at time k are obtained by averaging with the value at time $k-1$ in order to smooth their changes and to avoid unstable behaviors of the system.

5 A Working Example

In this section we present an example giving more concretely the flavor of our proposal. Even though this example has also the purpose of showing the effectiveness of the technique about the capability of selecting interesting peers which the query can be forwarded to, this latter issue is more deeply treated in Section 6. We consider a P2P system consisting of m peers P_i $(0 \leq i < m)$. Whenever a peer P joins the system, it obtains the IP of some other peers[2]. Such peers initialize the set of *adjacent* peers of P. Then, P contacts its adjacent peers in order

[2] In commercial applications such a set of peers is obtained by a server or by IP scanning.

Table 1. File distribution

file/peer	*rock*	*pop*	*rap*	*jazz*	*blues*
P_0	37	52	11	0	0
P_1	36	42	23	0	4
P_2	12	31	0	0	15
P_3	0	7	4	12	36
P_4	0	3	12	21	43
P_5	0	0	7	25	27

Table 2. Similarity among peers

simil.	P_0	P_1	P_2	P_3	P_4	P_5
P_0	1	0.89	0.74	0.18	0.18	0.12
P_1	0.89	1	0.81	0.25	0.24	0.19
P_2	0.74	0.81	1	0.38	0.31	0.26
P_3	0.18	0.25	0.38	1	0.93	0.72
P_4	0.18	0.24	0.31	0.93	1	0.93
P_5	0.12	0.19	0.26	0.72	0.93	1

both to receive their content descriptor and to compute similarity and expectation. The policy adopted in this example sets to 3 the number of "promising" peers which the query has to be forwarded to, at each hop. Such peers, called *contacts*, are determined according to the expectation-based strategy so far illustrated. For simplicity, we focus only on the communications between the peer P_0 and the 5 peers P_k (by assuming that $m > 5$), with $1 \leq k \leq 5$. We suppose the shared resources are audio files and each resource belongs to one of 5 music categories (that is the metadata category has 5 possible values). Table 1 reports the file distribution per peer. For example, the peer P_0 shares 37 *rock* songs, 52 *pop* songs and 11 *rap* songs. Table 2 contains all the similarity coefficients among peers. We analyze the evolution of the system starting from a situation in which P_0 has only P_1 and P_2 as adjacent peers and the parameters of the linear system are: $\beta_0 = 0.8$, $\alpha_{01} = 0.8$, $\alpha_{02} = 0.8$. The sensible values of α_{01} and α_{02} denote that P_0 have recently requested files shared from P_1 and P_2. Moreover, β_0 is also high as P_1 and P_2 are very similar to P_0. Accordingly with the maximum forward degree fixed to 3 in our example, we select at each iteration the three peers having the maximum expectation. Now we suppose P_0 searches a file not occurring either in P_1 or P_2. Thus, they propagate the query to other peers (including P_3). We suppose that P_3 sends the searched file to P_0. This causes that P_3 becomes a new adjacent peer of P_0. The above query produces the update of the parameters in the following way: $\beta_0 = \frac{0.8+0.18}{2} = 0.49$, $\alpha_{01} = \frac{0.8+0.25\cdot 0.8}{2} = 0.50$, $\alpha_{02} = \frac{0.8+0.38\cdot 0.8}{2} = 0.55$, $\alpha_{03} = 1$ according to the rules given in Section 3. The solution of the updated linear system, consisting of the expectations of P_0 w.r.t. P_1, P_2 and P_3 are: $E_{01} = 42,58\%$, $E_{02} = 41,10\%$, $E_{03} = 16,32\%$, expressed as a percentage. The expectation of P_0 w.r.t. P_3 is now comparable to those of P_0 w.r.t. P_1 and P_2. We observe that the similarity between P_0 and P_3 is low. At this point, suppose P_0 searches a new file by sending the query to P_1, P_2 and P_3 (remember the number of contacts is 3) and, this time, only the peer P_4 satisfies the request. Then, the updated parameters of the linear system (including also P_4) are: $\beta_0 = 0.26$, $\alpha_{01} = 0.31$, $\alpha_{02} = 0.36$, $\alpha_{03} = 0.97$, $\alpha_{04} = 1$. The new expectation values are: $E_{01} = 25,04\%$, $E_{02} = 25,75\%$, $E_{03} = 25,18\%$, $E_{04} = 24,03\%$. This result shows that the system is perceiving changing of user interests. In fact, in the last two steps, peers that are not much similar to P_0 (i.e., P_3 and P_4) have satisfied its queries, so that their expectation is increased, meaning that the peer P_0 is at moment interested in the contents of these peers.

Clearly, this increasing of expectation w.r.t. P_3 and P_4 has as a counterpart an initial decreasing of the expectations w.r.t. P_1 and P_2, that are peers which, despite their similarity, P_0 is currently reducing its interest in. However, among adjacent peers P_1, P_2, P_3 and P_4, only the first three are also contacts, so that P_0 will not send to P_4 the next query. Consider now a further iteration of the system, in which the peers P_4 and P_5 satisfy the request of P_0. In this case, the parameter of the linear system become: $\beta_0 = 0.20$, $\alpha_{01} = 0.19$, $\alpha_{02} = 0.24$, $\alpha_{03} = 0.93$, $\alpha_{04} = 1$, $\alpha_{05} = 1$, and the resulting expectations are: $E_{01} = 18,81\%$, $E_{02} = 19,84\%$, $E_{03} = 19,47\%$, $E_{04} = 25,00\%$, $E_{05} = 16,88\%$. In this situation, the peer P_1 is substituted by the peer P_4, according to our previous observation.

We observe also that the above example describes the system working during a transitory phase in which the user of P_0 is changing his interests. We can argue that, after a number of iterations, the content descriptor of P_0 will change becoming more similar to the last answering peers. Once this new "equilibrium interest" is reached, the similarity will become important (thanks to updating rules defined for the β coefficients), until new changes moves the user from this equilibrium. Our system thus shows the nice capability of capturing both equilibrium and transitory phases.

6 Experiments

In this section we describe the results of a number of experiments performed on a P2P simulator. We used a simulator in order to make feasible experiments with a large number of peers. It is widely accepted, in the context of P2P systems, that the results obtained by simulation are strongly significative for real-life applications.

We generated a network with 20 000 peers, in which 500 peers are simultaneously active and the number of neighbors per peer is 10. This experiment are representative for the population of 20 000 peers since in practice only around 5% of users are active at any given time [27] (see also [15] where experiments are conducted on the same number of peers).

We have considered a pre-built set of shared resources belonging to a universe of 100 domains, representing potential user's interests. Each user shares a certain number of resources belonging to i domains, where i follows a Zipf distribution [30], with $z = 1$. The Zipf distribution, originally used in linguistic field, is widely used for representing data distributions in various settings, like networks, Web systems, databases, etc. [11,3], and in particular P2P [22,15]. Here the Zipf distribution represents the fact that the most of users are interested in few domains.

We considered three query sets. The query set 1 is composed of files belonging to the domains the user is the most interested in. This query set models the case that user interests remain constant. The query set 2 is composed of files belonging to a new domain. This query set models the (stable) change of user interests to a new category. Finally, the query set 3 is obtained randomly combining the above two query set. This query set models an impulsive behavior of the user.

Each query has $TTL = 4$, that is the maximum number of hops of the query is 4. We called *Selection Size* (SS), the number of peers (among adjacent ones) the query is submitted to. We have performed experiments with several values for this parameter. In particular, we consider the case $SS = 10$ corresponding to the classic GNUTELLA flooding, and the cases in which $SS < 10$, where the adjacent peers to which forward the query are selected by exploiting our approach in one of the following variants:

1. method *Sim*, that considers only the similarity property, thus obtained by setting $\alpha = 1$ and $\beta = 1$ in the linear system;
2. method *Sim* & *Aging*, that uses both the similarity property and the aging coefficient, thus obtained by setting $\beta = 1$;
3. method *Exp* that adopts only the expectation property, thus obtained by setting α to 1;
4. method *Exp* & *Aging*, that fully implements our proposal.

In our experiments we analyze the traffic generated and the query success ratio versus the maximum contact degree of the peers with or without enabling expectations. In every case, we have tested the superiority of the method when expectations are enabled. Let us start with the description of our experiments.

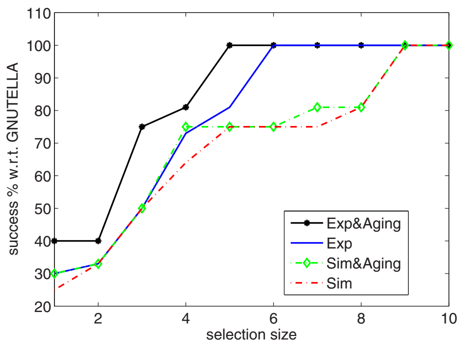


Fig. 4. Results for query set 1

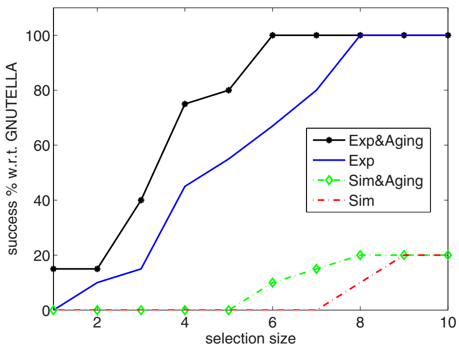


Fig. 5. Results for query set 2

Now we report the results of our experiments conducted by setting $\delta = 0.90$ (we recall that each peer sends to neighbors its current content-descriptor CD whenever the similarity between CD and the content-descriptor currently known by its neighbors is less than δ). First consider the query set 1. We recall that this query set models the case the user interests do not vary significantly over the time. In Figure 4 a graph displaying the number of successfully queries versus selection size, for all considered approaches, is reported. The figure shows that performances of methods *sim* (i.e., the similarity-based one) and *sim* & *aging* (i.e., the similarity/*alpha*-based one) are comparable. Of course, such approaches take advantages by the increasing of the selection size in a quasi-linear way. However, it is interesting to observe that their performances are lower than

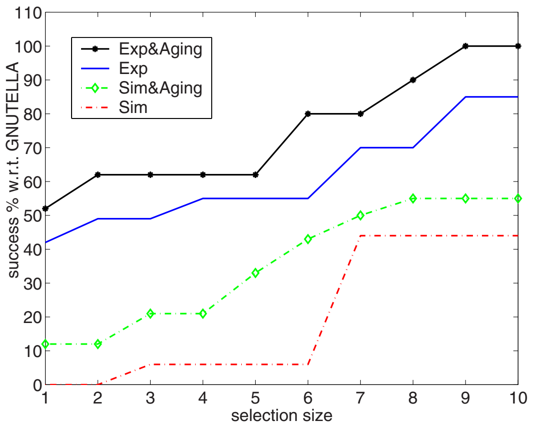


Fig. 6. Results for query set 3

the other two approaches. Expectation-based methods present the capability of rapidly "capturing" interesting peers. However the graph shows that considering the full approach (by taking into account both expectations and α coefficients) produces the best results. Consider now the query set 2. Recall that this query set models the (stable) change of user interests. Results are reported in Figure 5, where we can verify that the gap of performances between the expectation-based techniques and the similarity-based techniques is now significantly higher. This proves that the expectation-based methods are better than the others in terms of reactivity to interest changes, that is they are capable of adapting to new knowledge by using a small training query set. Note that the usage of the expectations produces a rapid success increase at small selection sizes, witnessing that whenever the selection size is just above a small threshold, the methods develop their full capabilities. In other words, above this threshold, the selection size does not work as a bottleneck against the full exploitation of properties implemented by expectations. From this perspective, similarity-based methods show a different behavior. They result significantly less sensitive w.r.t. benefits given by the increase of the selection size. This reflects the fact that similarities are properties inherently local. Again, observe that the best method is the full one, also in terms of sensitivity w.r.t. the selection size.

The last case considers the query set 3, modeling an impulsive behavior of the user. A problem which adaptive methods may suffer from, due to their capability of following user interests, is producing unstable behaviors. In this case, performances of these methods become dramatically worse whether the user behavior is impulsive. Experiments conducted on the query set 3 (as displayed in Figure 6) show that our method does not present the above drawback, since the full method shows still good performances definitely superior w.r.t. the other considered methods. This proves that as well as the linear system mechanism, are designed in such a way that a suitable degree of inertia avoids drastic changes and smoothes outliers occurring in the user behavior.

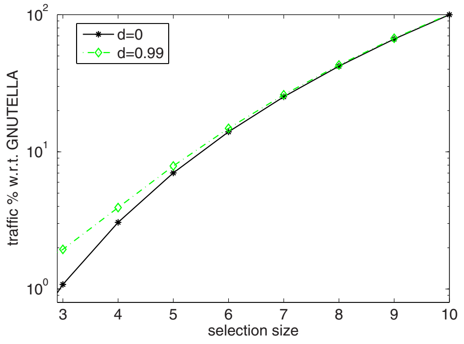


Fig. 7. Traffic measured

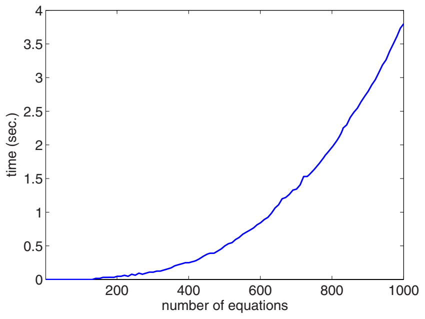


Fig. 8. System resolution time

In the next experiment we measured the traffic generated by varying selection size and by exploiting our proposal. The results are shown in Figure 7. We considered two values for the parameter δ that influences the traffic overhead due to the distribution of the similarity vector to the neighbors frequency of upload. Observe that according to the definition of δ, such a traffic is null when $\delta = 0$. The difference between the line for $\delta = 0.99$ and the line for $\delta = 0$ allows us to obtain an estimation of the traffic overhead due to the similarity vector sending, necessary to implement our proposal. Figure 7 shows that such an overhead decreases as the selection size increases and that, for realistic values of selection size, the overall traffic generated by applying our proposal is negligible. For example, setting the selection size to 7, that as shown by previous results gives us an high query success value, and $\delta = 0.99$, the traffic measured is about 25% of that one produced by GNUTELLA.

Finally, we have measured the time necessary for the computation of the expectation coefficients versus the number of adjacent peers. Experiments are performed by a Pentium IV with 1024MB. Results of such experiments are reported in Figure 8. Observe that when the number of adjacent peers is around 200, which is a value considerable high in real-life contexts, the time taken is very small (only 0.050 s).

7 Related Work

In this section we summarize several proposals addressing the information retrieval problem in P2P systems. In [17] the authors present some early measurements of a Cluster-based Architecture that uses a technique (Network-Aware Clustering) to group clients topologically close and under common administrative control. The introduction of one more hierarchy is aimed at scaling up query lookup and forwarding. [26] presents the Directed BFS technique, which relies on feedback mechanisms to intelligently choose which peer a message should be sent to. Neighbors that have provided quality results in the past will be chosen first, yet neighbors with high loads will be passed over, so that good peers do not become overloaded. In [9], message routing is improved with "routing indices", compact summaries of the content that can be reached via a link. With routing indices, nodes can quickly route queries to the peers that can respond, without wasting the resources of many peers who cannot. The authors of [24] study how to maximize the query answer rate of the entire system by setting the rate of query injection at each node. Concerning local and neighborhood properties, our paper may be related to [16], where the problem of identifying malicious peers in the network is faced. Indeed, the authors attack this problem by using the concept of *global trust value*, which is a value uniquely assigned to each peer of the network, reflecting the experiences of the other peers with that peer. The *global trust value* is obtained by a reputation system that aggregates the local trust values of all the users. [10] proposes a method to improve the search and scale performances in P2P systems where data is naturally clustered. The approach is based on *Semantic Overlay Networks* (SON), a flexible network organization in which nodes with semantically similar content are clustered together and queries are processed by identifying which SONs are better suited to answer it. They evaluate a classification hierarchy in order to select a small number of overlay networks whose nodes have a high number of hits, given a request. In [7] the authors propose the HELIOS framework to enable knowledge sharing and evolution considering a pure P2P system. The knowledge sharing and the evolution processes are based on peer ontologies, describing the knowledge of each peer, and on interactions among peers, allowing information search and knowledge acquisition, according to a pre-defined query model and semantic techniques for ontology matching. In [2] an agent-based P2P system architecture to support search for information through a flexible integration of semantic information is defined. Most of the papers cited above, and in particular [17,26,9,29,16,10], are close to our approach since they try to locate a peculiar group of peers in the network for better scaling queries. The only that, like our approach, uses global values computed by local values is [16]. However, it works in the different context of security in P2P systems, so that the meaning of local and global properties defined in it is quite different from our case. Our concept of similarity can be related to [18], where the authors propose a set of measures capturing the similarity of ontologies at two different levels, lexical and conceptual, focusing on "real-world ontologies". Finally, outside the context of P2P systems, our approach can be related to [5], where, in the setting of multi-agent systems, the usage of neighborhood-based properties is introduced. The approach used in [5] has some similarity with our one, but it cannot be used in P2P systems. Indeed, it works only in case of agent communities with a limited number of agents where a central agency, coordinating the cooperation, is available.

8 Conclusions

We have proposed a P2P strategy to increase the effectiveness of flooding-based information retrieval approaches. The strategy limits the number of peers which the query is forwarded to at each hop, allowing us to attack the problem of traffic explosion due to query propagation. The selection is done on the basis of semantic closeness among peers, measured through what we call the expectation, which relies both on local properties (i.e., similarity properties) and neighborhood-based ones. The effectiveness of the method is shown through a number of experiments that compare our approach with the standard flooding one (like GNUTELLA) at parity of retrieval success.

Acknowledgments

This work was partially supported by the research project QuaDRAnTIS, funded by the Italian Ministry of University and Research (MUR).

References

1. Brandt, A.: Multi-level adaptive solutions to boundary-value problems. Math. Comput. 31, 333–390 (1977)
2. Beneventano, D., Bergamaschi, S., Fergnani, A., Guerra, F., Vincini, M.: A peer-to-peer agent-based semantic search engine. In: Proceedings of the Eleventh Italian Symposium on Advanced Database Systems, pp. 367–378 (2003)
3. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web caching and zipf-like distributions: Evidence and implications. In: INFOCOM (1), pp. 126–134 (1999)
4. Buccafurri, F., Lax, G., Rosaci, D., Ursino, D.: A user behavior-based agent for improving web usage. In: Meersman, R., Tari, Z., et al. (eds.) CoopIS 2002, DOA 2002, and ODBASE 2002. LNCS, vol. 2519, pp. 1168–1185. Springer, Heidelberg (2002)
5. Buccafurri, F., Rosaci, D., Sarnè, G.M.L., Palopoli, L.: Spy: A multi-agent model yielding semantic properties. In: IAT-2001. Proceedings of the The Second Asia-Pacific Conference on Intelligent Agent Technology, pp. 44–53 (2001)
6. Bunch, J., Hopcroft, J.E.: Triangular factorization and inversion by fast matrix multiplication. Math. Comp. 28(125), 231–236 (1974)
7. Castano, S., Ferrara, A., Montanelli, S.: The helios framework for peer-based knowledge sharing and evolution. In: Proceedings of the Eleventh Italian Symposium on Advanced Database Systems, pp. 347–358 (2003)
8. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progression. Journal of Symbolic Computation 9(3), 251–280 (1990)
9. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: ICDCS 2002. Proceedings of the 22 nd International Conference on Distributed Computing Systems, p. 23. IEEE Computer Society, Los Alamitos (2002)
10. Crespo, A., Garcia-Molina, H.: Semantic overlay networks for p2p systems. In: tech. rep., Computer Science Department, Stanford University (2002)
11. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication, pp. 251–262. ACM Press, New York (1999)
12. Fensel, D.: Ontologies:A Silver Bullet for Knowledge Management and Electronic Commerce. Springer, Heidelberg (2001)
13. Freenet. The freenet home page, http://www.freenetproject.org

14. Gnutella, http://gnutella.wego.com
15. Kalnis, P., Ng, W.S., Ooi, B.C., Tan, K.-L.: Answering similarity queries in peer-to-peer networks. Inf. Syst. 31(1), 57–72 (2006)
16. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: WWW, pp. 640–651 (2003)
17. Krishnamurthy, B., Wang, J., Xie, Y.: Early measurements of a cluster-based architecture for p2p systems. In: Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement, pp. 105–109. ACM Press, New York (2001)
18. Maedche, A., Staab, S.: Measuring similarity between ontologies (2002)
19. Napster, http://www.napster.com
20. Press, W.H., Teukolsky, S.A.: Multigrid methods for boundary value problems. Computers in Physics, 514–519 (1991)
21. Reif, J.H.: Efficient approximate solution of sparse linear systems. Computers Math. Applic. 36(9), 37–58 (1998)
22. Sen, S., Wang, J.: Analyzing peer-to-peer traffic across large networks. IEEE/ACM Trans. Netw. 12(2), 219–232 (2004)
23. Strassen, V.: Gaussian elimination is not optimal. Numerishe Mathematik 13, 354–356 (1969)
24. Sun, Q., Daswani, N., Garcia-Molina, H.: Maximizing remote work in flooding-based peer-to-peer systems. Computer Networks 50(10), 1583–1598 (2006)
25. Tamma, V., Wooldridge, M., Blacoe, I., Dickinson, I.: An ontology based approach to automated negotiation. In: Alonso, E., Kudenko, D., Kazakov, D. (eds.) AAMAS 2002. LNCS (LNAI), vol. 2636, Springer, Heidelberg (2003)
26. Yang, B., Garcia-Molina, H.: Improving search in peer-to-peer networks. In: ICDCS 2002. Proceedings of the 22 nd International Conference on Distributed Computing Systems, p. 5. IEEE Computer Society, Los Alamitos (2002)
27. Yang, B., Garcia-Molina, H.: Comparing hybrid peer-to-peer systems. The VLDB Journal, 561–570 (September 2001)
28. Zadeh, L.A.: Similarity relations and fuzzy orderings. Information Sciences 3, 177–200 (1971)
29. Zeinalipour-Yazti, D., Kalogeraki, V., Gunopulos, D.: Exploiting locality for scalable information retrieval in peer-to-peer networks. Information Systems 30, 277–298 (2005)
30. Zipf, G.K. (ed.): Human behaviour and the principle of least effort. Addison-Wesley, Reading, Mass (1949)

Improving the Dependability of Prefix-Based Routing in DHTs

Sabina Serbu, Peter Kropf, and Pascal Felber

University of Neuchâtel, CH-2009, Neuchâtel, Switzerland
{sabina.serbu, peter.kropf, pascal.felber}@unine.ch

Abstract. Under frequent node arrival and departure (churn) in an overlay network structure, the problem of preserving accessibility is addressed by maintaining valid entries in the routing tables towards nodes that are alive. However, if the system fails to replace the entries of dead nodes with entries of live nodes in the routing tables soon enough, requests may fail. In such cases, mechanisms to route around failures are required to increase the tolerance to node failures.

Existing Distributed Hash Tables (DHTs) overlays include extensions to provide fault tolerance when looking up keys, however, these are often insufficient. We analyze the case of greedy routing, often preferred for its simplicity, but with limited dependability even when extensions are applied.

The main idea is that fault tolerance aspects need to be dealt with already at design time of the overlay. We thus propose a simple overlay that offers support for alternative paths, and we create a routing strategy which takes advantage of all these paths to route the requests, while keeping maintenance cost low. Experimental evaluation demonstrates that our approach provides an excellent resilience to failures.

Keywords: fault tolerance, reliability, DHT, routing.

1 Introduction

Dependability concerns many properties of a system, such as scalability, reliability, security, data integrity, availability, routing or fault tolerance. These properties are generally dealt with according to the system's architecture. In this paper we address the *accessibility* of the stored data, focusing on the routing and fault tolerance issues in structured P2P systems.

Consistent information about the nodes in the system is crucial for effective operation and it is essential that a peer-to-peer system allows for fault tolerance. Thus, a silent node departure shall not turn the whole or part of the system inoperable. Though information (data) may disappear from the system when a node silently departs, the routing functions that it assumed shall be taken over by other peers alive. This calls for some system management and maintenance functions as far as the overlay network organization and associated routing functions are concerned. Clearly there is a trade-off between the costs of such maintenance and the effectiveness of the routing achieved, which depends of course on the properties of the particular overlay structure.

The main trend in existing P2P systems is to essentially focus on preserving stored data and preserving consistency of the network structure, ignoring *fault-tolerant access* to the data under churn, i.e., frequent node arrival and departure. To provide data accessibility in a system under churn, both a fault-tolerant infrastructure and fault-tolerant routing need to be taken into account. In the following, we present these aspects and later in Section 2 we show some of the existing methods providing for fault tolerance.

Distributed hash-tables (DHTs) use specialized placement algorithms to assign responsibility for each object to peers as well as "directed search" protocols to efficiently locate objects. With regard to the infrastructure aspect, they rely on a large variety of different structures, such as rings, multidimensional spaces, hypercubes or other types of graphs. One notable difference between these structures is the *degree* that each node in the overlay has, which, in this context, is the number of neighbors with which a node maintains continuous contact for supporting the routing mechanism. A *constant node degree* assures low maintenance costs for the entries in the routing tables, costs related to the control traffic that is required to check for the state of the neighbors and to set a new node for an entry that is found to contain a dead node. Unfortunately, this also means that they do not offer a significant tolerance to faults. Examples include de Bruijn-based overlays [1], Viceroy [2] or CAN [3]. Other DHTs use a *logarithmic node degree*, such as Chord [4], Pastry [5], Tapestry [6] or Kademlia [7]. These systems show higher costs for maintaining the routing tables compared to the systems that use a constant node degree. Nevertheless, they can use alternative entries when an entry fails, which provides a good start base towards a *fault-tolerant infrastructure*. This is the reason why, in our research, we are focusing on this type of overlays.

An overlay infrastructure needs to be able to recover from failures by replacing entries of dead nodes with entries of live nodes in the routing tables. To update such an entry in the routing table, one must find a node that would fit at that entry. Since it is costly and mostly impossible to keep all routing tables entries always populated with live nodes, these updates are made periodically: at each time interval, maintenance requests are issued and the routing tables are updated. As a consequence, this still leaves a time window when entries may refer to dead nodes. Because routing table entries become often invalid under churn, the system has to additionally provide *fault-tolerant routing* by finding alternative routes to forward the requests towards the destination. As in [8], we say that the overlay routing is *dependable* if a request reaches its destination.

To discuss fault-tolerant routing, we illustrate in Section 3 the case of Chord-like DHTs which use *greedy routing*, one of the most-known and widely-used routing algorithm. Greedy routing is simple: at each routing step, the request is directed towards a node as close as possible to the destination. This strategy provides fast lookup because the number of hops is minimized. In case of node failure, greedy routing algorithms typically apply a "route around" strategy by using a lower entry from the routing table if the normally chosen entry contains a dead node. Experimental results have confirmed that greedy routing under node failures is an unreliable strategy with respect to fault tolerance and routing dependability. Indeed, the advantage of getting as close as possible to the destination at each routing step (i.e., going as far as possible from the source) becomes a disadvantage under node failures, as this strategy exploits only a small part of the possible paths. At each routing step, the number of possible paths towards destination is heavily decreasing, which drastically diminishes the chances of finding a valid path to destination.

Following the analysis of standard greedy routing, we propose an overlay structure and a routing scheme to provide a high degree of fault tolerance, while still keeping maintenance costs low:

– the *hypeer* overlay is a logarithmic node degree DHT with a structure that approximates a hypercube.
– the FT-routing strategy is an efficient routing scheme that allows multiple options in the selection of a next node in the request path to provide fault tolerance in case of churn.

The *hypeer* overlay offers the choice between many redundant paths, which is needed in a fault tolerant system. As all other DHTs, it uses an identifier space where the nodes and the keys obtain IDs in a form of a sequence of binary digits. To route towards a node responsible for the requested key, several intermediate nodes are traversed such that the digits from the source identifier are successively replaced by the digits of the key identifier. Our proposed structure is loosely based on a hypercube. This offers the possibility of treating the digits in any order when routing from source to destination, which tunes the number of redundant paths. The redundant paths considerably enhance fault tolerance.

The rest of the paper is organized as follows. In Section 2 we discuss related work on hypercubes and fault-tolerance support. Section 3 further details the motivation of our work. In Section 4 we present our system: its structure, routing strategies and functionality. In Section 5 we present the experiments conducted and discuss the results obtained. Then, we conclude in Section 6.

2 Related Work

In this section we present related work for hypercubes, which represents the structure that offers the highest choice for alternative paths, and then some of the existing solutions for fault-tolerance with respect to the infrastructure and the routing strategies. Note that we do not deal with any security aspects, such as trusted nodes or trusted information (this is well detailed in [9]).

2.1 Hypercube-Based DHTs

There are several DHTs that use the hypercube structure in order to provide alternative paths. This allows for fault tolerance, however with a penalty of increasing the complexity of the overlay maintenance.

The eQuus [10] system has a topology of a partial hypercube. Each vertex represents a clique, i.e., a group of nodes that are close in terms of a proximity metric. The lookup procedure is similar to Pastry [5], with the main difference that each entry represents a clique and not a single node. The nodes in a clique share the same ID and keys. Thus, fault tolerance is mostly treated from the point of view of data availability, and not to achieve routing fault tolerance.

Schlosser *et al.* [11] present HyperCuP, a hypercube structure that is built as peers join the system. A node keeps its neighbors on a per-dimension basis, and it might have the same node as neighbor in two or more dimensions, if no other suitable node has been found. When joining the system, a new node contacts an existing random node which will become its new neighbor in a dimension of its choice. The strongest point of this solution is the idea of the hypercube construction, however a node may become responsible of too many vertices of the hypercube, thus its failure may severely affect the routing. Moreover, the usage of broadcast messages to all or a part of the dimensions of the hypercube may become too costly in terms of number of messages.

In [12], Alvarez *et al.* propose to increase the number of path connections through the use of a hypercube structure. Each node has an identifier and a mask that indicates the ID space that the node is responsible for. The routing algorithm can be either proactive, assuring a specific route to each node based on a tree distribution of the IDs, or reactive by creating on demand a route and keep it for a certain period of time. The usage of route creation makes this solution seem more adequate for systems where churn rates are rather low.

2.2 Fault Tolerance with Other Structures

In order to achieve fault tolerance, the resource discovery mechanism described in [13] is based on an arrangement of multiple Chord rings, each one responsible for a keyword. However, the system relays on a super ring which contains pointers to each Chord ring. This solution aims for fault tolerance, however the super ring is a critical point of failure.

Wepiwé *et al.* [14] propose a concentric multi-ring overlay for high reliability, where the nodes on a given inner ring form a de Bruijn graph. This overlay assumes knowledge about the reliability of the nodes, which normally is not a constant in any system.

2.3 Extensions for Fault-Tolerance

Backup Nodes. The easiest and most widely adopted solution to deal with dead nodes in routing tables is the addition of *backup nodes* (redundant links). The best-known examples are systems like Chord [4] or Pastry [5]. In Chord, each node maintains a list of a fixed number of successors on the ring. When an entry has failed, a lower entry is used. For the lowest entries, the list of successors may be used. Lam *et al.* [15] propose the K-consistent networks. Each node keeps always K nodes at each entry in its routing table. Whenever a node from the routing table fails to respond, a repair mechanism tries to find a new suitable node for the same entry. This type of solutions is obviously limited by the number of backup nodes used. A high number of backup nodes means a higher number of alternative paths, and so a higher probability of success. However, the backup nodes need also maintenance, so the disadvantage is seen in the additional costs imposed by maintaining more node entries in the routing tables.

Reducing the number of dead entries. Castro *et al.* [8] use a different approach to the ones mentioned so far, proposing techniques to detect node failures and repair routes. They apply this solution in MSPastry, a particular implementation of Pastry. These techniques successfully decrease the number of dead entries in the routing tables, however there is no solution to completely eliminate them, which means that there is still the need for routing around failures.

Replication. Replication is one of the most simple solutions for fault-tolerance, where several replicas of the same object are placed at different nodes. These nodes are either chosen uniformly in the identifier space, in the neighborhood of the destination, or using a replica function [9]. Replication can be easily applied as a complementary solution to any fault-tolerant infrastructure or routing solution.

2.4 Fault-Tolerant Routing

For dependable routing under failures, Aspnes *et al.* [16,17] propose two extensions for greedy routing. When a node cannot find another node that is closer to the destination than itself, it can use either *random re-route* (random choice of another node to forward the request to), or *backtracking* (sending back the request to the previous node in the request path by keeping track of some visited nodes). These two extensions provide reasonable results, however, they still exploit only a small number of possible paths.

Backtracking is however a good technique to enlarge the number of alternative paths, but it is not well exploited when used with greedy routing. A request that gets close to the destination, but is forced to use backtracking, would do small back hops, which means that the gained number of alternative paths remains small.

Another possibility to increase the request success rate is *redundant routing* (as it is called in [9], or *parallel routing* as in [18]). In this case, several copies of the same request are sent towards the same destination through different paths. In [9], for fault-tolerance, such copies are sent from the source to a set of its neighbors towards the nodes that own replicas of the requested object and following different paths. Independently of the choice for the next hops of the paths, when applying redundant routing, more requests are sent in the system, so more processing is required at the nodes. This, obviously, increases the costs considerably.

In contrast to the existing solutions for fault tolerance, we aim to improve *dependability* by allowing at each routing step to consider the maximum number of possible alternative paths, even if no failure has been detected yet.

3 Motivation

To provide a high level of fault tolerance, we consider it necessary to take into account both the overlay and the routing strategy.

3.1 On the Dependability of Greedy Routing

Many DHTs use greedy routing to forward the requests because of its simplicity. This strategy gives good results in terms of path length, but it has limitations in the number of paths it can exploit. To get from source to destination, greedy routing adjusts the bits from left to right when the request is forwarded to the next hop in the request path. This strategy generally leads to *path convergence*: the last hops of most requests for a certain destination pass through only a small set of nodes, which are mostly the preceding neighbors. Under a failure-free operation, these nodes are likely to be overloaded if the destination is very popular. Furthermore, if one of them fails, the traffic will be severely affected.

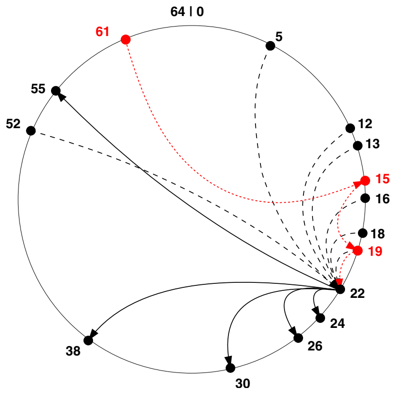


Fig. 1. Example of Chord with an identifier space of $2^m = 64$

Systems such as Chord [4] or Pastry [5] suffer from these limitations of greedy routing. A graphical representation of a Chord example is shown in Figure 1. In Chord, each node and object has a m-bit identifier on a 2^m ring, obtained by respectively hashing the IP address and the name. The objects are mapped to their subsequent node on the ring. For routing purposes, each node has a routing table with m entries, each entry i pointing towards the first node on the ring at a distance of minimum 2^i, where $i = 0..m - 1$. Conversely, each node is in the routing table of other nodes, so it has incoming links from these nodes. In the example of Figure 1, 15 (out of 40) nodes are shown on a 2^6 ring. The incoming links of node 22 are shown with dashed dark lines, and its outgoing links are shown with solid dark lines. While each of the outgoing links points to nodes at a distance close to a power of 2 away, the distance from the incoming link nodes is less predictable. Each request is forwarded by greedy routing, following always a clockwise path, as for example the request going from node 61 to node 22 in three hops (the dashed grey line).

Figure 2 shows, in percentages, the cumulative distribution function (CDF) of the number of requests that are received per incoming link in a Chord-like system with no node failures. In this experiment, the identifiers are mapped on $m = 15$ bits. The system has 10,000 nodes and 20,000 keys, with 200,000 requests uniformly issued. As can be seen on the left-hand side of the graph, most of the traffic is received from the incoming links with small distances, which limits the possibility for redundant paths. More than 80% of the traffic is received from the incoming links at 2^i away, where $i \leq 4$. Note that the first entries of the routing table (small values of i) may point to the same node because of the inter-node distance. Thus, the predecessors of a node are critical nodes because they bear the majority of the traffic for that node. If such a node fails, the rate of request success drastically decreases.

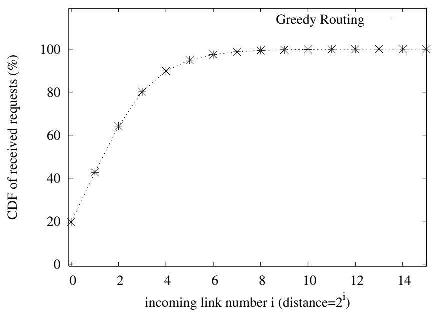


Fig. 2. The Cumulative Distribution Function (CDF) of the percentage of received requests per incoming link (2^i) in a Chord system using greedy routing

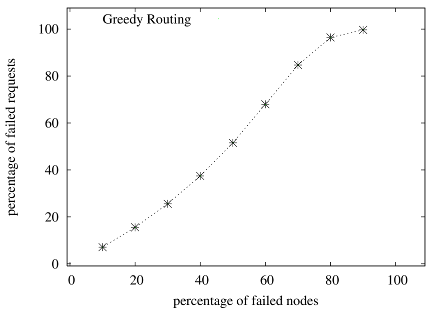


Fig. 3. Failure rate of greedy routing. The high percent of failures shows that greedy routing does not exploit the redundant paths

In the experiment under failures, whenever a routing table entry refers to a failed node, a lower entry is used instead. Figure 3 shows the percentage of failed requests when varying the percentage of failed nodes. The graph shows that this strategy is not dependable: when half of the nodes fail, half of the requests also fail.

The main reason for this poor performance is that alternative paths are not exploited. At each node, the request is sent as close as possible to the destination. This means that the distance between a next hop node n_{nh} and the destination n_d of a request is minimal. Unfortunately, it also means that the number of possible alternative paths is minimal once the request has reached n_{nh}. As an example, a request in Figure 1 goes from node $n_s = 61$ to node $n_d = 22$, by going through nodes 15 and then 19, according to greedy routing. The possible alternative paths go through each of the incoming links of n_d: 5, 12, 13, 16, 18 and 19. At n_s, greedy routing chooses $n_{nh} = 15$. When this node is reached, the request (which always travels clockwise) may pass only through the incoming links 16, 18 and 19. If these nodes fail, the request will also fail to reach its destination, even though alternative paths from n_s through the incoming links of n_d, nodes 5, 12, or 13, would be valid.

Another aspect of this kind of overlays is that the outgoing links of a node are not exactly at 2^i distances, so the request does not necessarily follow 2^i jumps. This fact prevents from applying a deterministic routing strategy to exploit other valid paths.

All these observations uncover the mismatch between the goal of providing fault tolerance and the means used for lookup with greedy routing. The extensions for fault-tolerance may give good results, however, if better support for exploiting alternative paths is already considered at overlay design time, even better results may be obtained.

3.2 The Hypercube Structure Approach

Based on the above observations, we seek for a structure that best provides alternative paths, where the routing algorithm is able to process in any order the digits from the source identifier to match that of the destination. In such cases, the total number of available paths is $m!$, where m is the number of digits in the identifier sequence. After i bits have been treated, the number of available paths is $(m - i)!$. Such a routing procedure that exploits alternative paths is well achievable in *hypercube structures*. Some hypercube based systems were already described in Section 2.

In a hypercube architecture with N nodes, each node has a $O(\log N)$ node degree, where the number of neighbors is equal to the number of dimensions. Each neighbor has an identifier that differs by exactly one digit.

The problem of using the hypercube for an overlay is that it requires complex protocols to deal with churn. When new peers join, the number of dimensions has to be increased. Conversely, when peers leave the system, the overlay has to treat the dimension split problem. Because of these problems and the high costs their treatment induces, deterministically constructed hypercubes are not suitable to provide fault tolerance under high churn.

However, to take advantage of the (structural) availability of alternative paths for routing in a hypercube, while avoiding the drawbacks of increased maintenance costs for rearranging the structure in case of churn, one could use an approximation of a hypercube that is built probabilistically.

In our study, we seek to achieve this by assigning non-random IDs when peers join. The idea is to assign to a new node n_b an ID that is exactly at a power of 2 away from an existing random node n_a. Thus, a hypercube vertex is set to n_b and an edge is created from n_a to n_b. As more peers join, new dimensions of the hypercube fork spontaneously by populating its edges and vertexes. In terms of the overlay structure, after node n_b joins the system, node n_a will have n_b at entry i of its routing table, and n_b will be at exactly 2^i away from n_a.

Figure 5 illustrates this structure, which we call a pseudo-hypercube. It has an almost deterministic node placement, and as a consequence, we can route almost deterministically. Greedy routing is still supported, but we can also use non-greedy routing algorithms that are more efficient for fault tolerance, as presented in the next section.

4 *hypeer* Design

Our design goal is based on the discussion in the previous sections. We propose *hypeer*, a DHT with a ring structure embedded in a directed pseudo-hypercube supporting several routing strategies.

The nodes and the keys have IDs in an identifier space of length 2^m, where m is the number of bits in the identifier sequence. The responsibility for a key is given to the first node that follows the key on the ring (as it is the case in Chord).

In contrast to the common method of using a hash function to map the nodes on the ring, we choose to assign the node identifiers in a way to approximate a hypercube structure by trying to maintain an even inter-node distance equal to a power of 2 despite churn. We call *inter-node distance* the distance between a node and its predecessor.

Each node has a link to its predecessor and successor on the ring (see Figure 4). The routing table of a node contains entries which are, with a high probability, at exactly 2^i away, with i from 0 to $m - 1$. These are the neighbors of the node in the hypercube. The links between a node and its neighbors are shown with arrows in Figure 5. The

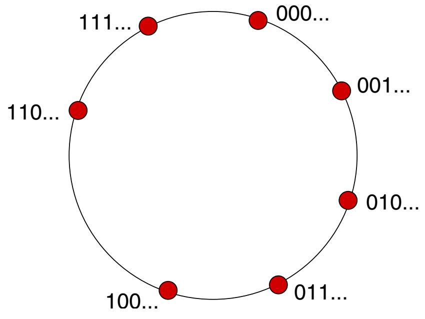


Fig. 4. The ring structure

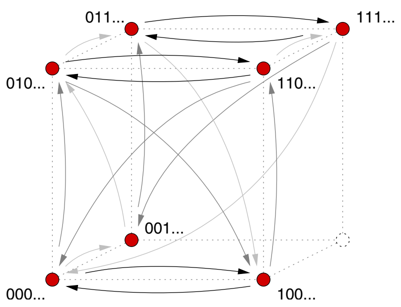


Fig. 5. The Hypercube structure

nodes 2^{m-1} away from each other have bidirectional links (they are in “diametrical opposition” on the ring) and are denoted by black arrows. The gray arrows are the other links in the hypercube approximation.

The routing procedure is based on replacing the bits from the source ID by the bits of the destination ID, i.e., bit i from the identifier sequence is replaced through a jump to a node at 2^i away. Our hypercube structure provides in most cases links towards nodes that are exactly at 2^i away. However, in some cases, and mostly for small values of i, links may point to nodes that are not exactly at 2^i away. Thus, a jump to such a node would affect also some of the bits at the right-hand side of the replaced bit. One or more of the bits on the left side may be affected only when changing a bit from 1 to 0, because of the resulting carry over. Thus, in *hypeer* there are not only links towards nodes with only one bit changed as in a proper hypercube. To construct a proper hypercube, the i^{th} link should point counterclockwise if the i^{th} bit is set, and clockwise if it is not set, which would create double links between nodes.

A fault tolerant routing strategy must provide a large number of alternative paths even when the request is close to the destination (*close* means a small number of digits that are different in the source and destination identification sequences). As a consequence, we propose a solution where the request follows small steps in the beginning of the routing path (where the low-order digits are treated) and then longer steps (treating higher-order digits). It is clear that when no failures occur, the strategy gives similar results to greedy routing, because the number of hops to route a request is of the order $O(\log N)$, assuming that we can treat the bits in any order. However, when failures do occur, this strategy exploits a much higher number of possible paths, as the requests are routed around failures with a higher probability.

In short, we are applying simple modifications to Chord-like systems. Chord cannot easily exploit redundant paths because of its non-determinism in node placement that does not permit treating digits in any order, so we are fixing this by adding some determinism in the placement of the nodes. This is obviously advantageous for the routing strategy due to the control of node position. Besides fault tolerant routing, this structure can also adopt a routing strategy to balance the traffic load on the path towards a popular destination.

4.1 *hypeer* Overlay

Our system has the IDs assigned deterministically, with the purpose of creating a hypercube-like structure. The main idea is to maintain 2^i links between the nodes and implicitly an inter-node distance of a power of 2 in order to take advantage of the redundant paths of the hypercube for routing.

We start dealing with fault tolerance at design time of our overlay. As explained before, we do not use a hash function to map the nodes on the ring. When a new node arrives in the system, it sends a request to a random key (e.g., obtained by hashing the IP address of the new node). The node responsible for that key adds the new node as a 2^i neighbor. A new vertex and the edge between the existing node and the new node are thus populated. The joining procedure is shown in Algorithm 1.

Algorithm 1. Node n_a receives a Join Request

```
0:  {First decide whether $n_a$ or its predecessor assigns the ID}
1:  **if** $dist(pred(n_a), n_a) > dist(n_a, succ(n_a))$ **then**
2:      Send the request to $pred(n_a)$
3:  **else**
4:      {Check each entry $x$ of the routing table $RT$, starting from the highest entry}
5:      $entry \leftarrow x$, where $RT(x) \neq n_a + 2^x$
6:      **if** $entry$ found **then**
7:          Respond with $n_b \leftarrow n_a + 2^x$
8:          $RT(x) \leftarrow n_a + 2^x$
9:      **else**
10:         Send request to $pred(n_a)$
11:     **end if**
12: **end if**
```

A new node n_b that wants to join the system has to issue a join request towards a random ID. The request is routed in the system until it arrives at node n_a which is responsible for that ID.

Node n_a consults its routing table to find the entry that has not yet a node that is exactly at a power of 2 away, by starting with the highest entries pointing to the furthest away nodes (lines 4-5). We thus give priority for perfect 2^i edges between nodes which are at long distances from one another on the ring (high i), because these are not affected by the inter-node distance that can vary. Node n_b will be assigned with an ID equal to (n_a + 2^x), where x is the entry found (lines 7-8).

In the case that n_a has at each entry x a node at exactly 2^x away, the joining request is sent to its predecessor (lines 9-10).

Node n_a is chosen randomly, as the associated ID was chosen randomly. As an optimization, with the objective to not partition the ID space into small pieces, n_a checks if its predecessor $pred(n_a)$ is further away than its successor $succ(n_a)$, and in such a case, it asks $pred(n_a)$ to assign the ID (lines 0-2). Of course, more complicated schemes to choose this node may be used for a better approximation of the hypercube. However, randomness gives good results, as we show later in Section 5. Besides creating the hypercube-like structure, this scheme also assures the uniformity of the distribution of the node IDs in the identifier space.

After n_b obtains its ID, it will start populating its routing table. Node n_b issues requests towards IDs that are 2^i away from itself, where i goes from 0 to $m-1$. Node n_a

will wait for a short period of time before updating its routing table with n_b to allow n_b to create its own routing table.

To detect node joins and departures, the routing tables are periodically updated, in the classic way of issuing requests to the ID that each entry should accommodate, i.e, a node n sends requests to all $n + 2^i$, with i from m to 0.

4.2 Routing Strategies

First we present our fault tolerant routing strategy that may be used with the *hypeer* overlay, and then we discuss other routing strategies, pursuing other goals such as traffic load balancing.

Fault-Tolerant Routing (FT-routing)

Algorithm. For fault tolerance, we want to take advantage of all available incoming links of a destination, starting from the source node and going clockwise towards the destination on the ring.

The idea of the routing algorithm is to reach the nodes that have direct 2^i links towards the destination. These nodes are, with a high probability, the nodes that are responsible for the IDs $keyId - 2^i$, where $keyId$ is the requested key and i goes from 0 to $t - 1$. The value of t depends on the distance between the source node and the key, where $keyId - 2^t$ represents the furthest incoming link of the destination that is between the source and the destination on the ring.

The routing algorithm at node n_x is presented in Algorithm 2. We show later in this subsection how we deal with node failures.

Algorithm 2. FT routing algorithm

```
1:  upon receive lookup($T$, $key$) at node $n_x$
2:  {Receive request}
3:  if $pred(n_x) < key \leq n_x$ then
4:      {Node $n_x$ is responsible for $key$: success}
5:  else if $T \leq pred(n_x) < key$ then
6:      {Went too far: send to the predecessor}
7:      Send lookup($T$, $key$) to $pred(n_x)$
8:  else
9:      {Compute next hop}
10:     FT-route($key$, $key$)
11: end if
12:
13: function FT-route($T$, $key$)
14: if $\exists n_k \in RT$ s.t. $n_k - range < T \leq n_k$ then
15:     {This node is probably responsible for $T$}
16:     Send lookup($T$, $key$) to $n_k$
17: else
18:     {Compute new $T$}
19:     $T \leftarrow T - 2^i$, where $i$ is max in $T - 2^i > n_x$
20:     FT-route($T$, $key$)
21: end if
```

Upon receiving a lookup request, node n_x directly responds to the request if it is responsible for the key (lines 1-4) and the request path ends here.

At each node n_x from the request path, a next node on the request path towards the requested key needs to be found. We use a recursive function, which we call FT-route(T, key) with parameters T and key. It terminates when it has found a node to forward the request to. T is a target (an identifier) that we aim to reach on the ring from n_x in a single hop. If n_x is not able to directly send the request to T, a new target T is set and the function is called recursively. In the first call of FT-route(T, key) at any node, T is set to the destination key, and then T decreases in the subsequent recursive calls with the largest power of 2 possible such that T is still between the source and the destination on the ring. With each call, the power of 2 decreases and the target gets closer to the source.

In the following, we say that a node is responsible for an identifier, if the identifier is between the node and its predecessor on the ring (in the same way as we map keys to nodes). Furthermore, we say that a node is *probably* responsible for an identifier when the identifier is between the node and an estimation of the position of its predecessor on the ring. When we are not aware of the position of the predecessor (as it is the case for the predecessors of the nodes in the routing table) we use an estimation of the inter-node distance.

Node n_x needs to find a node n_k from its routing table which is probably responsible for the target T. The condition at line 14 is satisfied, if T lies on the ring between $n_k - range$ (the assumed predecessor node of n_k, where $range$ is the estimation of the inter-node distance) and n_k. Since the nodes are uniformly distributed on the ring and because a large enough estimation of the inter-node distance shall be used, we set $range$ as the maximum between the inter-node distance of n_x and the inter-node distance of its successor. This can be easily computed since n_x knows its predecessor and successor IDs. Other values might be used as well, for example the average of the inter-node distance of all the nodes that the request passed through, which of course would add this value to the request message.

The chosen estimation works well in most cases due to the fact that the inter-node distance is the same for the majority of the nodes, as we will show later in Section 5. However, in some cases it happens that the request is sent further away than the node responsible for the target. The request is then sent to the responsible node by using one or more (but only few) hops through predecessor links (lines 5-7). To treat such cases, the target itself is added to the request message when forwarding the request to a probably responsible node for a target (line 16).

If n_x did not find a suitable node in its routing table to forward the request towards T, it will set a new target, which is 2^i before the current target on the ring. For the fault tolerance goal, we choose i as the maximum value between 0 and $m - 1$ such that the new target is still after n_x on the ring (lines 18-20). This assignment for recursive calls creates a virtual path where each hop is a power of 2, and moreover, the powers of 2 increase with each hop.

Note that the algorithm provides flexibility for the order of fixing the digits. If we choose i as the minimum value, this leads to a form of greedy routing. Moreover, if i is chosen randomly, this leads to a routing strategy that balances the load on the incoming links of the destination. This latter strategy is further detailed in the next subsection.

Treating Failures. In the following, we show how the above algorithm deals with the failures of nodes in the system.

If no entry from the routing table of n_x points to a live node (from those who point toward nodes before the destination on the ring), we say that a dead-end has been reached. Then, the request is backtracked to the previous node in the request path. The same happens when the request needs to be sent to the predecessor (line 5 of Algorithm 2) and the predecessor is down or a dead-end. To be able to use backtracking, before forwarding a request, a node adds its ID to the request message.

However, if the suitable entry is down or a dead-end, but other nodes from the routing table are not, n_x chooses another entry as follows. If n_x has found a node that is probably responsible for the target, but it is down or a dead-end, it will choose a node at a higher entry (starting from the following entry, going up), in order to ”jump” over that target. If no node is found suitable from the higher entries (i.e., alive, not a dead-end and before the destination on the ring), a smaller entry (starting from the previous entry, and decreasing) is used. We choose first the higher entries and then the smaller ones because we want the request to quickly by-pass the faulty target, and not to increase unnecessarily the path length in the attempt to reach the same target that seems to belong to a dead node.

Without loss of generality, we do not consider failures of the destination node, as we treat fault tolerance from the routing point of view. If the destination node is down, its keys are lost anyway. Any request for such a key would then result in a non successful lookup.

Other Routing Strategies

LB-routing and GR-routing. Alternative routing strategies to the FT-routing algorithm described above include the following:

- *random-order routing (LB-routing)*, where the digits are replaced in a random order, but not necessarily independently. The fault tolerance is not as high as for the FT-routing, but considers traffic load balancing on the incoming links.
- *greedy routing (GR-routing)*, where the digits are treated from left to right. Our overlay supports also greedy routing, however, neither fault tolerance nor load balancing are expected.

Besides fault-tolerance, another advantage of alternative paths is to reduce the traffic load of the last nodes on the paths to a popular key. This is the reason for including LB-routing as a routing strategy in *hypeer*. LB-routing implements the same algorithm as FT-routing (presented in Algorithm 2), with the difference that at line 19, the new target T is chosen randomly. However, to maintain the same virtual path at each node, the targets have to be chosen the same at each node. Otherwise, setting different targets at each hop might increase significantly the path length.

The results for LB-routing have also been included in the experiments presented in Section 5 as a compromise between GR-routing and FT-routing.

To additionally improve fault tolerance, several copies of the same request may be forwarded using a different routing strategy. This technique is called redundant routing[9] or parallel routing[18].

5 Evaluation

In this section we analyze *hypeer*, a structure with a uniform partitioned space, and FT-routing in *hypeer*, as a good routing strategy with or without failures. To present the results, we compare FT-routing with GR-routing and LB-routing.

5.1 Overlay Structure

Our approach for assigning node IDs to new nodes ensures that they are at a distance of 2^i from some existing node. Further, on expectation, nodes are uniformly distributed in the identifier space. To validate this claim, we have simulated 10,000 node arrivals in an identifier space of 2^m, $m = 20$ and then computed the distribution of inter-node distances.

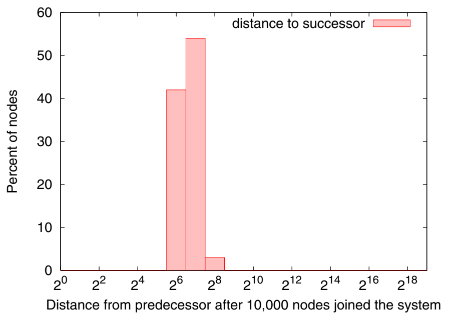


Fig. 6. Inter-node distance for 10,000 nodes in an identifier space of 2^{20}

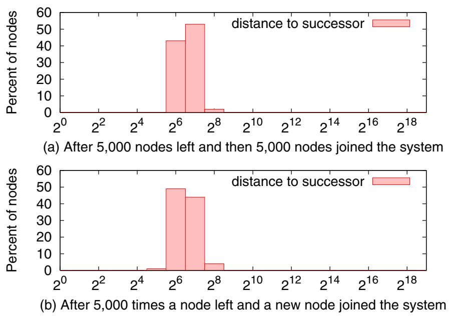
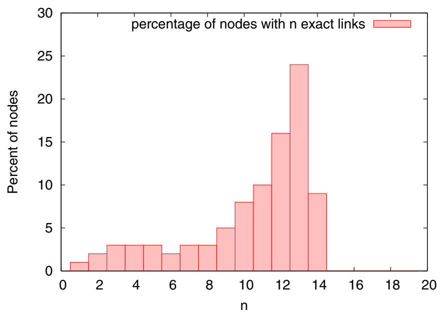


Fig. 7. Inter-node distance under churn

As can be seen in Figure 6, among all 20 possible distance values, most of the nodes (roughly 95%) have an inter-node distance of either 2^6 or 2^7. Having two values with consecutive exponent for inter-node distance is expected, because of the continuous change in the hypercube structure caused by the new nodes arrival.

With this overlay we further analyzed the inter-node distance when dealing with churn. The results of two scenarios are depicted in Figure 7: (a) a scenario where 5,000 nodes leave and then 5,000 nodes join, and alternatively (b) a scenario with 5,000 successions of a leave followed by a join. Here, we observe that churn is only slightly affecting the overlay: the inter-node distances remain almost the same.

Next, we analyzed the number of outgoing links that are at exactly 2^i away from the current node, as shown in Figure 8. The maximum number of different outgoing links is 14 (out of m=20), because at least the first 6 entries of the routing tables point to the successor node, as a direct consequence of the inter-node distance of minimally 2^6. This explains the 0-percentage of nodes having a number of outgoing links larger or equal to 15 (the right-hand side of the graph), and also the smaller percentage for 14 outgoing links. The main observation is that the graph has an increasing tendance: a



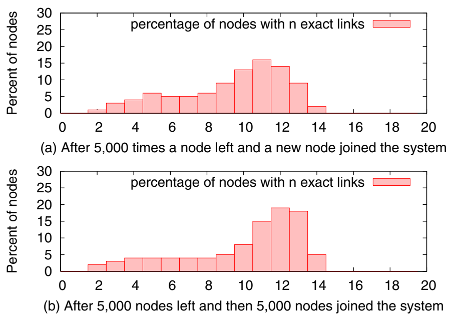


Fig. 8. Percent of nodes with the same number of outgoing links at exactly powers of 2 away, for 10,000 nodes in an identifier space of 2^{20}

Fig. 9. Percent of nodes with the same number of outgoing links at exactly powers of 2 away under churn

higher percentage of the nodes have a higher number of outgoing links towards nodes at 2^i away. This means that the hypercube edges at 2^{m-1} are populated first, and then the lower ones. The same observation holds when analyzing the incoming links.

As in the inter-node distance analysis, churn has only a light effect. Figure 9 shows the number of outgoing links with the same two scenarios as for Figure 7 for an overlay with 10,000 nodes. Again, the 0-percentage on the right-hand side of the graphs is justified by the inter-node distance, which has not been chopped by churn. Moreover, the graphs continue to show the increasing tendency.

We can thus conclude that our join algorithm produces a structure that is quite uniform and regular, which is key to deterministically locate redundant paths and route around failures.

5.2 Routing Under Failure-Free Operation

The fact that greedy routing results in big steps at the beginning of the request path limits the number of alternative paths if one node in the request path fails. Conversely, FT-routing proceeds by small steps in the beginning and larger ones in the end. This allows for a larger number of alternative paths until the destination is reached. Intuitively, FT-routing can be seen as striving to "keep all options open" while greedy routing would rather proceed "rushing blindly". This more careful behavior of FT-routing, which is key to ensuring fault-tolerant routing, is analyzed next.

In the following experiments, we consider a system with 10,000 nodes and 20,000 objects in a space of 2^m, $m = 15$, where we issue 200,000 requests. When using greedy routing, a request is sent to the highest node entry smaller than or equal to the requested key.

We have first run experiments in ideal settings without node failures to analyze the path lengths of FT-routing and to compare it against those of greedy routing (GR-routing). We have also included the analysis of LB-routing. Table 1 shows the average and the variance for the path lengths obtained with the three routing strategies in *hypeer*. We note that the results obtained do not differ significantly, which indicates that FT-routing and

LB-routing perform well under failure-free operation. The higher average and variance of FT-routing can be explained by the fact that it can better locate and exploit very short paths, but at the same time some paths are longer than on average because of the incomplete hypercube embedding.

Table 1. Statistics for path length with no failures

Routing Strategy	Average	Variance
GR-routing	7.27	3.5
LB-routing	7.46	4.5
FT-routing	8.66	24.8

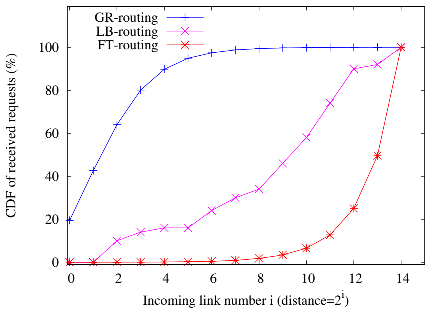


Fig. 10. Comparison between the percentage of received requests per incoming link (2^i) using GR-routing, LB-routing and FT-routing with no failures

We have then analyzed the average load on the incoming links of the destinations of the issued requests, by repeating the experiment shown in Figure 2 with FT-routing (the results for greedy routing are shown again for comparison purposes). Additionally, we present the results for random routing. Figure 10 shows that, with FT-routing, the incoming links that are used the most are the furthest-away ones from the destination. The reason is that the requests are sent from the source to the closest node that has a direct link to the destination. This trend contrasts with GR-routing, which essentially relies upon close links to destination. LB-routing, where the next hop is chosen at random among the nodes that have a link to the destination, represents a compromise between GR-routing and FT-routing and balances the load on all incoming links.

We have also analyzed the case where the request has to backtrack along predecessors (because of an inappropriate estimation of the inter-node distance). Under the same experiment with 200,000 requests, the real owner of the target is only one or at most two steps away. Moreover, it happens with only a small probability of 6%. This means that the estimation of the inter-node distance performs well.

5.3 Routing Upon Failure

To validate the robustness of our routing algorithm, we have run experiments when a given proportion of random nodes fail simultaneously. This adverse scenario simulates correlated failures, e.g., network partitions. We have observed how effective the three routing strategies (GR-routing, LB-routing and FT-routing) are at reaching a given key right after the node failures occur, i.e., before the routing tables have been repaired.

Failure Rates. We deploy two types of experiments under failures. First, we analyze the results for the base algorithms, and next we apply backtracking to each of them. In all experiments we vary the proportion p of failed nodes from 10% to 90%.

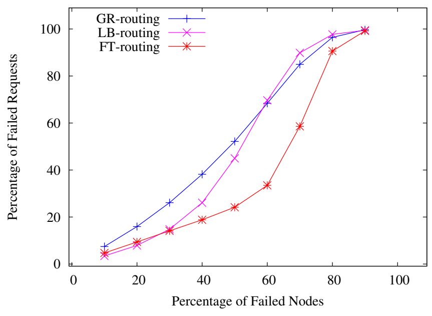


Fig. 11. Failure rates: Comparison between GR-routing, LB-routing and FT-routing, without using backtracking

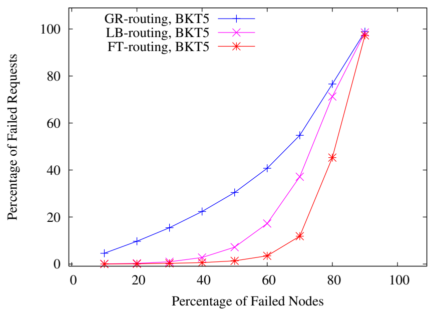


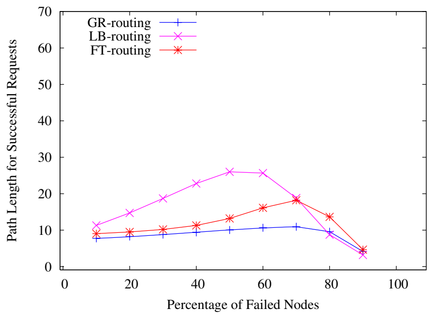
Fig. 12. Failure rates: Comparison between GR-routing, LB-routing and FT-routing, using a backtrack chain of 5 nodes

Figure 11 shows the comparison between the failure rates of the three routing strategies without using backtracking. For up to 60% node failure, FT-routing has a percentage of failed requests equal to half of the one obtained by GR-routing. From 70% on, the results are still better for FT-routing. As expected, LB-routing has a percentage of failed requests between GR and FT routing. Not surprisingly, at high percentage of node failures, the results are similar for all three routing strategies.

Figure 12 shows the same comparison, but this time all routing strategies use backtracking. In the experiments, we have used a backtrack chain of 5 nodes. The results are obviously better for each of the three routing strategies, again FT-routing obtaining the best results, and LB-routing being in the middle. However, we observe that their results differ much more this time. Backtracking acts much better with FT-routing than with GR-routing, because, when the request is close to the destination and has to backtrack, the jumps back are larger in the case of FT-routing (un-fixing the high order bits), so a larger number of redundant paths can be exploited thereafter. For FT-routing, only a few requests are lost for failure rates of up to p=70%. For instance, with 50% node failure, FT-routing reaches almost always the destination (only 1.4% of the requests fail) while GR-routing can only deliver one third of the requests (30.45%). The low percentage of failed requests for even high rates of failures demonstrates the high dependability of FT-routing.

We have compared our results with the results obtained by Aspnes *et al.* in [17]. The authors showed that backtracking is a good solution to exploit alternative paths, and moreover they applied heuristics to the routing table maintenance. Their results are good, however we obtain better results with FT-routing for up to 70% node failures.

Average Path Length. The average path lengths for the two types of experiments (without and with backtracking) are shown in Figures 13 and respectively 14.



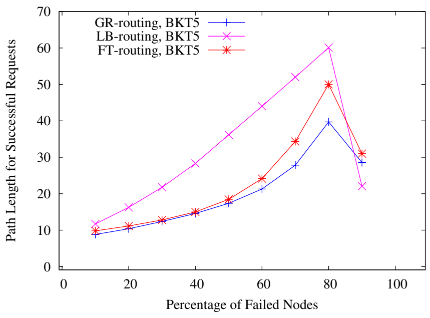


Fig. 13. Average Path length: Comparison between GR-routing, LB-routing and FT-routing under failures, without using backtracking

Fig. 14. Average Path length: Comparison between GR-routing, LB-routing and FT-routing under failures, using backtracking with 5 nodes

When backtracking is not used (Figure 13), as expected, the path length for GR-routing is the smallest. The increase in the path length of FT-routing is justified by the additional requests (compared to GR-routing) that are successful. For LB-routing, the path length is the longest. This is mostly caused by the random choice of the bits to be treated, which can lead in some cases to too small order bits to be treated and thus implying hops that are smaller than the inter-node distance.

Backtracking significantly increases the path lengths of all routing strategies (Figure 14), since the jumps back are also counted. The results for FT-routing and GR-routing are getting closer. This can be justified by the requests that backtrack when they are close to the destination. In such cases, GR-routing makes smaller hops than FT-routing, and so it needs more hops to go back to a certain node.

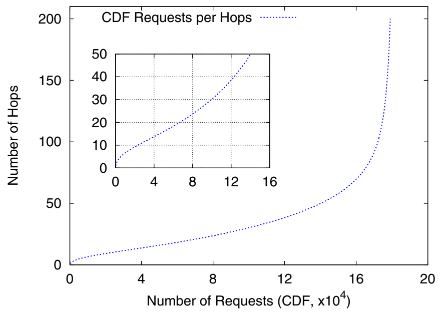


Fig. 15. CDF of the number of requests per number of hops, where 200,000 requests were issued with a maximum acceptable path length of 200 hops and of 50 hops in the inner graph

Table 2. Percentage of failed requests when using a backtrack chain of 5 nodes and different maximum acceptable path length

Routing	Req Failures	Path avg
FT, max 200 hops	11.86%	34.3
FT, max 50 hops	29.50%	23.0
GR, max 200 hops	54.78%	27.8

In all experiments, we have set the maximum acceptable path length to 200 hops. Figure 15 shows the cumulative distribution function (CDF) of the number of requests per number of hops under 70% node failures and using a backtrack sequence of 5 nodes (the total number of issued requests is 200,000). In this case, 11.86% of the requests failed (as shown earlier in Figure 12). The average path length is 34.3 hops. There is only a small part of the requests that have very long paths (notice the very quick increase in the path length at the right-hand side of the graph). Thus, we choose to limit the maximum path length to 50 hops (see inner graph), so the requests with path length larger than 50 hops are considered as failed. In this case, the average path length decreases to 23 hops (as expected, since the increase in path length until 50 hops is almost linear), but of course the percentage of failed requests increases. It becomes 29.5%, which is still smaller than in case of GR-routing. Table 2 summarizes these results.

The Choice for Redundant Routing. At very high failure rates (80% and 90%) experiments showed that the request failures tend to be independent of the proximity between the source and the destination. However, the request failures experienced by FT-routing for node failures of up to 70% were identified as being mostly due to the proximity between the source and the destination, which severely limits the number of redundant paths. In such cases, our algorithm could be extended to also search for paths that traverse nodes outside the range between source and the destination, i.e., by initially moving away from the destination. For example, for small distances between the source node and the destination, redundant routing could be applied, by sending a request using FT-routing, and another request through the highest routing table entry available, and then apply FT-routing to take advantage of all incoming links of that particular destination. This could also be an alternative solution to using backtracking.

6 Conclusions

The analysis on the fault-tolerant infrastructures and routing strategies shows that the support for fault-tolerance cannot be an afterthought when willing to provide a high fault tolerance at low costs. Greedy routing is simple, however for fault-tolerance, it lacks of dependability.

Consequently, we designed both the infrastructure and the routing strategy of our overlay with the goal to offer the support for fault tolerance. We applied simple modifications to Chord-like systems. Chord cannot easily exploit redundant paths because of its non-determinism in node placement that does not permit treating digits in any order, so we are fixing this by adding some determinism in the placement of the nodes. This means that we may control the position of the nodes on the ring, which is obviously advantageous for the routing strategy. In contrast to the common method of using a hash function to map the nodes on the ring, we approximate a hypercube structure by trying to maintain an even inter-node distance equal to a power of 2 despite churn. Then, we are also modifying the routing protocol to exploit alternative paths by taking into account all possible incoming links of the destination starting from the source node. The rate of request success is much higher, and the maintenance cost remains low, since no additional structures that need to be maintained are required.

Our experiments clearly demonstrate that the FT-routing algorithm, combined with uniform space partitioning that allows us to route deterministically via multiple paths, provide an excellent resilience to failures.

References

1. Kaashoek, M.F., Karger, D.R.: Koorde: A simple degree-optimal distributed hash table. In: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems, pp. 323–336 (2003)
2. Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: A scalable and dynamic emulation of the butterfly. In: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing, pp. 183–192. ACM Press, New York (2002)
3. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: Proceedings of ACM SIGCOMM, pp. 161–172. ACM Press, New York (2001)
4. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of ACM SIGCOMM, pp. 149–160. ACM Press, New York (2001)
5. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
6. Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A., Kubiatowicz, J.: Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications 22(1), 41–53 (2004)
7. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Proceedings of the 1st International Workshop on Peer-to-Peer Systems, pp. 53–65 (2002)
8. Castro, M., Costa, M., Rowstron, A.: Performance and dependability of structured peer-to-peer overlays. In: DSN2004. Proc. 2004 International Conference on Dependable Systems and Networks, pp. 9–19 (2004)
9. Castro, M., Drushel, P., Ganesh, A., Rowstron, A., Wallach, D.: Secure routing for structured peer-to-peer overlay networks. In: OSDI2002. Proc. 5th Symposium on Operating Systems Design and Implementation, pp. 299–314 (2002)
10. Locher, T., Schmid, S., Watternhofer, R.: eQuus: A provably robust and locality-aware peer-to-peer system. In: Proceedings of the 6th International Conference on Peer-to-Peer Computing, pp. 3–11 (2006)
11. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: Hypercup – hypercubes, ontologies and efficient search on p2p networks. In: Moro, G., Koubarakis, M. (eds.) AP2PC 2002. LNCS (LNAI), vol. 2530, pp. 112–124. Springer, Heidelberg (2003)
12. Alvarez-Hamelin, J.I., Viana, A.C., Amorim, M.D.: DHT-based functionalities using hypercubes. In: Proceedings of World Computer Congress IFIP WCC, vol. 212, pp. 157–176 (2006)
13. Salter, J., Antonopoulos, N.: An efficient fault tolerant approach to resource discovery in p2p networks. Technical Report CS-04-02, University of Surrey Guildford (2004)
14. Wepiwé, G., Simeonov, P.L.: A concentric multi-ring overlay for highly reliable p2p networks. In: NCA, pp. 83–90 (2005)
15. Lam, S.S., Liu, H.: Failure recovery for structured p2p networks: Protocol design and performance evaluation. In: Proceedings of ACM SIGMETRICS - Performace, pp. 199–210. ACM Press, New York (2004)
16. Aspnes, J., Diamadi, Z., Shah, G.: Fault-tolerant routing in peer-to-peer systems. In: PODC2002. Proceedings 21st ACM Symp. on Principles of Distributed Computing, pp. 223–232. ACM Press, New York (2002)
17. Aspnes, J., Diamadi, Z., Shah, G.: Greedy routing in peer-to-peer systems. extended version of Fault-tolerant routing in peer-to-peer systems (2006)
18. Oh, E., Chen, J.: Parallel routing in hypercube networks with faulty nodes. In: ICPADS, pp. 338–345 (2001)

Social Topology Analyzed

Njål T. Borch[1], Anders Andersen[2], and Lars K. Vognild[1]

[1] Norut AS, Tromsø, Norway
[2] Department of Computer Science, University of Tromsø, Norway

Abstract. It is an aspiring trend that Internet users not only consume, but also produce and share content. This leads to content of great diversity. Personalized navigation is an approach to correctly discover interesting content for the user. This navigation can be based on both search and recommendation systems.

The Socialized.Net is a social Peer-to-Peer network infrastructure supporting personalized navigation. In this paper, a data trace of a popular file sharing site is analyzed and shown to have semantically close users. The data trace is further used for simulations in *The Socialized.Net*. Evidence is given that a fully distributed social network can be created based on traffic analysis. This can provide a powerful platform for personalized content navigation.

Keywords: Social topology, p2p search, personalization, integrity, performance.

1 Introduction

Internet users are moving away from being pure consumers. They are increasingly also becoming producers of content. While the content creation is performed in a distributed manner, centralized distribution system is often required. Weblogs and wikis are examples of centralized services produced and consumed by the end users.

Peer-to-Peer (P2P) networks naturally map onto a world of fairly equal peers. P2P can allow users to decouple themselves from the central distributor. It allows them to freely cooperate, assist and provide services to each other. Such decentralized solutions could improve the user experience by removing some of the limitations of centralized solutions. A few examples of such limitations are bandwidth usage, data storage, freedom of speech and monetary cost.

While P2P networks can provide a powerful and suitable platform for resource sharing, there are a number of issues to resolve. First, an efficient and easy way to locate resources is of vital importance. This would likely be a combination of searching, browsing and recommendations. Second, the peers must have an incentive to share resources with each other. With little or no incentive, peers are less likely to donate resources to the network. This limits the total resources of the network, limiting its usefulness. Third, the network integrity must be high, ensuring the authenticity of located resources. For example, if a game application is expected a virus is not acceptable.

In the next section, related work is presented. In section 3, *The Socialized.Net* is introduced. Section 4 presents a real-world data trace and how these data are used for simulations. A set of performed simulations with *The Socialized.Net* are described in section 5. Section 6 presents a set of analysis of both data traces and the simulated topologies. Finally concluding remarks and future work is presented in 7.

2 Related Work

In order to perform efficient searches, most centralized solutions use inverted indexes [1]. These indexes allow fast lookups, providing results for a large number of users within fractions of a second. In order to provide relevant and high quality results, these search engines typically pre-calculate a rating for each resource. Pagerank [2] is an example of such a rating system, giving each web page a certain rank. Due to their complexity, these ranks cannot be calculated dynamically for each search, but must be pre-calculated by the system. This makes it difficult to provide personalized search results. In effect, the more popular resources will always be preferred, which might not be appropriate given the variance in taste of the users.

Another approach to searching is to analyze usage patterns. This approach is often used in recommendation systems. One strength is that such systems can work fairly well even though the resources themselves cannot be categorized or analyzed automatically. The way "Last.FM" recommends music based on similar user's taste is one example. Another is to recommend a set of head phones when buying a portable music player, as can be experienced on Amazon.

During the development of decentralized content distribution, a myriad of Peer-to-Peer networks have been created. Gnutella [3] was the first popular fully decentralized P2P network with incorporated search facilities. The flooding algorithm used for searching would forward queries to all nodes, thereby giving fairly good guarantees that available resources would be found. This algorithm does however not scale very well [4]. Newer versions of Gnutella is based on a hierarchical structure [5], which to a certain degree improves the scalability of the system.

The early P2P file sharing network Napster utilized a central server to provide the index. This server thus became a single point of failure, and the network failed to operate when the server was stopped [6]. Most large P2P file-sharing networks utilize servers to locate resources. In order to provide a more stable infrastructure, these networks can change the set of servers dynamically. A failing server can therefore be replaced by other servers without inconveniencing the users.

Distributed hash table (DHT) based P2P networks provide a distributed inverted index [7]. DHTs work by using a hash function to locate a responsible node within the topology. This allows them to provide very efficient key lookup [8,9,10]. The keys, for example file names, must however be known in advance in order for these networks to function properly. This leads to a dependency of some other mechanism in order to search for resources. DHTs also have issues with incentive. The hashing of keys means that nodes in a DHT network are typically put in charge of keys they do not have any interest in. The nodes are thus less likely to spend enough resources to provide a good service.

The popular BitTorrent protocol does not incorporate any search facilities. This requires users to share meta-information about resources ("torrent" key files) through other channels. This sharing is typically done via web pages. The BitTorrent protocol does however have a fairly successful incentive algorithm, namely the "Tit-for-tat" algorithm [11]. The basic idea is that the faster a peer can deliver data, the more data it will receive. When resources are scarce, this algorithm will focus resources to the best connected nodes. These nodes are in turn likely the best candidates to provide the resources to yet more nodes. The Tribler project [12] tries to solve the distribution and navigation of torrent files by providing a recommendation system within BitTorrent client software.

Semantic routing is a different approach to Peer-to-Peer networks. In semantic routing algorithms, only peers likely to provide relevant information are queried [13,14]. The number of messages in the system is thus kept low. Semantic networks solve incentive issues by keeping the resources at the nodes who find them interesting. It is of course possible to use a swarming protocol like BitTorrent to perform the actual data transfer. While efficient, semantic networks are prone to malicious hosts. Malicious hosts are peers that knowingly provide false information or resources. For example, they might send a virus instead of the requested resource. In semantic networks, malicious nodes might also be able to control their own location in the topology of the overlay network. This can be done by pretending to have an interest or providing some popular resources. Their strategic position can later be exploited in order to perform attacks. Malicious nodes could therefore be a larger issue in semantic networks than in hash based or centralized systems.

3 The Socialized.Net

The Socialized.Net [15] is a fully distributed Peer-to-Peer search infrastructure. It is based on a semantic routing protocol utilizing semantic similarity and a locally calculated node preference.

Semantic knowledge of nodes is gathered by observing the traffic in the network. For each neighbor, a node weights each observed keyword by how often it is seen. These keyword statistics are the base for the semantic similarity of nodes. When routing a query, a score is calculated for each neighbor as the sum of the intersection of keywords of the query and the neighbor. The semantic routing is thus performed dynamically for each query.

Statistics about the behavior of neighbors is also gathered. These statistics are used to calculate a set of ratios. The ratios are only used locally, and as such are not communicated between nodes.

- *Contribution Ratio*
  describes whether a neighbor contributes to the network by providing replies as opposed to only sending queries.
- *IGot* ratio
  describes whether the node has resources of interest for the neighbor.
- *YouGot* ratio
  the reverse of *IGot* ratio, describing whether the neighbor has resources of interest for the node.
- *Relay ratio*
  seeks to prioritize direct contact between nodes by penalizing neighbors that require relaying of large numbers of messages.
- *Bogus ratio*
  measures the ratio of how many uninteresting announcements and replies are received from a neighbor.

The ratios are summarized in a locally calculated, subjective *preference* [16]. Nodes will gossip about their preference with their neighbors, spreading reputations. This gossip is piggy-backed on liveness messages, thereby limiting the impact on network usage. Preference and reputation is calculated periodically. Direct interaction is also allowed, enabling users or applications to change the preference for a node. For example, an application can ban nodes that provide files with mismatching file types and extensions.

The preference and reputation together allows each node to rate all its neighbors. When routing queries, this rating is added to the dynamic semantic score of each neighbor. This allows nodes that statistically has been seen to be contributing and often provide good results to be prioritized. Similarly, misbehaving nodes can be penalized.

4 Filelist.org Analysis

Social networks are graphs that can exhibit small world properties [17]. This phenomena states that social networks is an efficient way to find short paths through graphs based largely on local information. The success is however dependent on incentive [18] and the presence of "long links" [19]. A long link is defined as a binding between two nodes that are topologically placed a substantial distance away from each other.

In semantic networks, the small world phenomena is exploited by using similarity in interests in order to group nodes. Locating a resource is thus question of finding a group with corresponding interests. An experiment analyzing the Gnutella network for illegal pornography [20] demonstrated tight grouping between nodes sharing such content. It is believed that similar grouping also exists for other subjects.

In order to verify grouping on interests, we have performed data analysis on trace data gathered from the popular "FileList.org" BitTorrent file sharing site. "FileList.org" has a limit of 100 000 registered users (beginning of 2006).

The data trace was performed by the University of Delft, the Netherlands. The trace was performed by crawling the "FileList.org" website repeatedly. Lists of available files were refreshed every 200 seconds, current download statistics approximately every 10 minutes.

The analysis was performed on three months of trace data from January 2006 through March 2006. The data set contains 87 129 distinct nodes sharing a total of 3 275 resources described by 3 290 distinct keywords. Each resource is described by 1 to 12 keywords, with an average of 3.7.

The data has been analyzed with respect to grouping nodes by keywords. By doing this, we can verify the assumption that nodes can be grouped based on semantic distance. As nodes are likely to keep an interest for longer than it takes to download a file, we do not require two nodes to download a file at the same time in order to share interest in the file. The number of shared keywords between each node was counted and the 25 top neighbors of each node were stored.

Due to the computational power and memory demands of simulating such a vast amount of nodes, we did a random selection of 10% of these nodes for simulation. In the figures presented below and in the simulation, these 8 713 nodes were used. These nodes shared the same resources as the complete set of nodes. This is due to the relatively small number of resources in the community. In fact, during the three months of monitoring, the average number of new resources was only 36 per day.

The number of nodes having at least one resource in common were then counted. The results are shown in figure 1, and show that a substantial amount of common resources are present. This is expected, as the number of resources is low in comparison with the number of nodes. More than 90% of the nodes have more than 250 nodes sharing common resources.

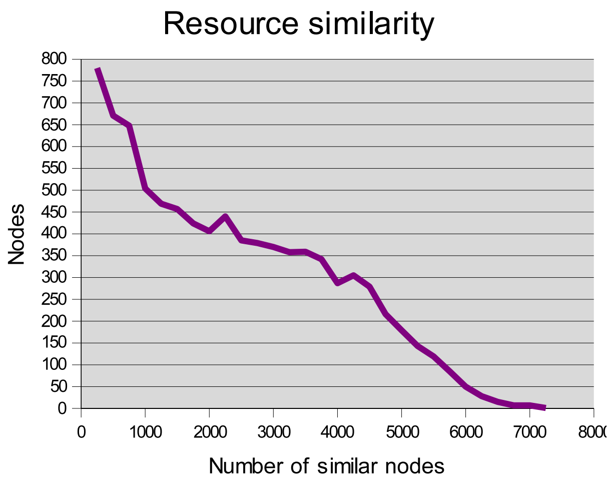


Fig. 1. Resource distribution

5 Simulation

In order to get a good understanding of how *The Socialized.Net* performs, the trace data described above was used in simulations. Only information about "which nodes shared what resources" was extracted from the trace, no actual measure of node activity was available. For the simulation, a set of nodes was therefore selected to be malicious and another set of nodes to be free-riders. This was done to verify whether *The Socialized.Net* is able to efficiently handle malicious and free-riding nodes. The malicious nodes should receive the worst ratings in order to limit their impact on the network. Free-riders should also receive bad ratings as they are less likely to be of use. All nodes with ID less than 10,000 were defined as malicious. This is a set of 51 nodes (0.6% of the simulated nodes) that will reply to any query with resources that does not match the query itself. All nodes with ID of over 800 000 is deemed a free-rider, and will as such not reply to any query. This is approximately 2 600 nodes (30% of the simulated nodes). For the FileList.Org community, these numbers are likely smaller due to the internal rating and subsequent banning of free-riders and inactive users. The selection does however allow us to monitor how the different routing protocols handle malicious and free-riding nodes.

A custom cycle based simulator was written in order to limit the memory footprint while maintaining complete neighbor lists for all nodes. The simulator was largely written in SQL, utilizing a database to efficiently maintain the state of the network. This was done due to the large amount of gathered state in the network. During each cycle, each 20th node generated a query with a set of 3 keywords selected from the node's associated resources. All nodes are equally active, so after 20 cycles all nodes would create and transmit a query. In order to bootstrap the network, a central node was used. *The Socialized.Net* bootstraps in a similar way, even though possibly using more than one bootstrapping node[1].

Three different routing algorithms were tested: random routing, semantic routing and social routing. Flooding (sending to all neighbors) was not possible due to massive resource usage.

A random routing protocol would select 6 random neighbors. A purely semantic routing algorithm was also implemented. This algorithm would select the semantically closest neighbors based on the query being routed, as described earlier. 3 to 6 neighbors were selected for routing. More neighbors were selected if they were believed to be semantically closer. This basic semantic routing does not detect free-riders or malicious nodes. Replies that are invalid are discarded, and both free-riders and malicious nodes will slowly be integrated into the infrastructure.

Social routing was implemented by also including preference and reputation. This routing is expected to perform similarly to the pure semantic routing albeit with malicious nodes rated lower. Free-riders should also be penalized, although they should be preferred over malicious nodes.

[1] *The Socialized.Net* also uses local node discovery to allow nodes on local networks to introduce each other into the network. This reduces the load on the bootstrapping nodes as the network grows in popularity.

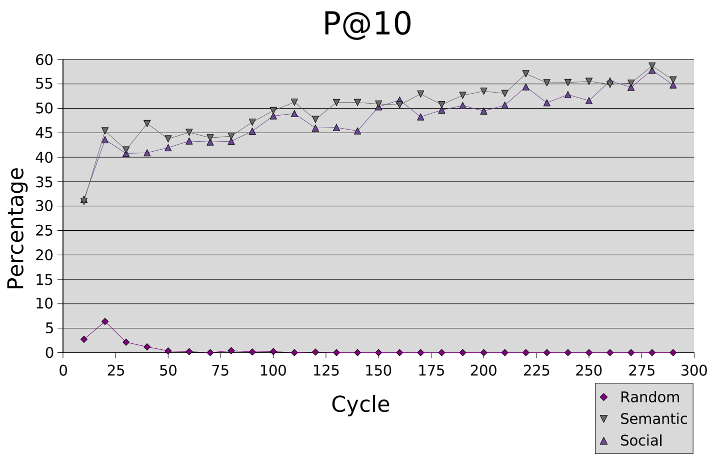


Fig. 2. P@10

The simulations were run for 300 cycles. Every 10th cycle, a set of test searches were performed. The same set of nodes were used for all searches. A query would be created by selecting a set of random keywords within the interest field of the node. A full P2P search would then be performed, without updating the simulator state. The set of "relevant documents" for the query was then selected from the database. The precision of the top 10 resources returned by a search could then be calculated as

$$p@10 = \frac{RetrievedDocuments \times 100}{RelevantDocuments}$$

As illustrated by figure 2, the two semantic protocols follow each other closely. They are very quickly able to provide 45% relevant results. As we have relatively few resources in the system, the relevant set for each query is quite small. This means that if only a very few resources are missing from the reply set, the value of P@10 will drop significantly. The random routing performs very badly. A possible explanation can be that as the neighbor cache grow, the random routing selects between more neighbors. As we only perform searches with relatively few nodes, we should see peaks when the routing was "lucky" and actually hit one or more nodes with matching resources.

The semantic algorithms seek to be efficient by limiting the amount of forwarded queries. In order to verify the protocols, we calculate the *Query efficiency* by dividing the total number of queries in the network over the average retrieval rate. This indicates how efficient each query is in triggering good replies. As shown in figure 3, the random routing fares very badly. The retrieval rate is poor at the same time as the number of queries in the system is high. The semantic protocols have similar efficiency, with social routing performing slightly better due to fewer messages being transmitted.

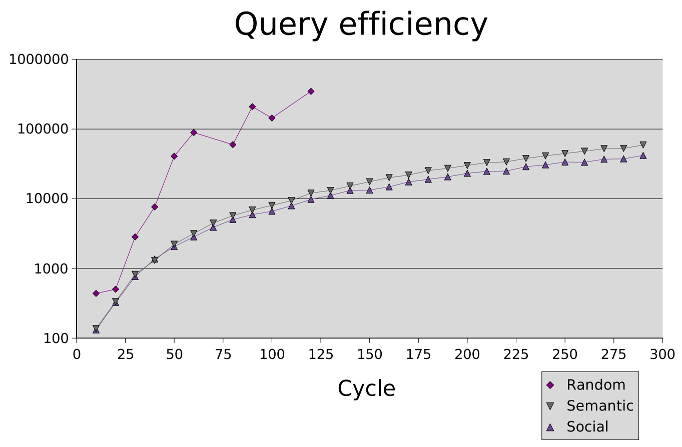


Fig. 3. Efficiency of queries, logarithmic function. Lower is better.

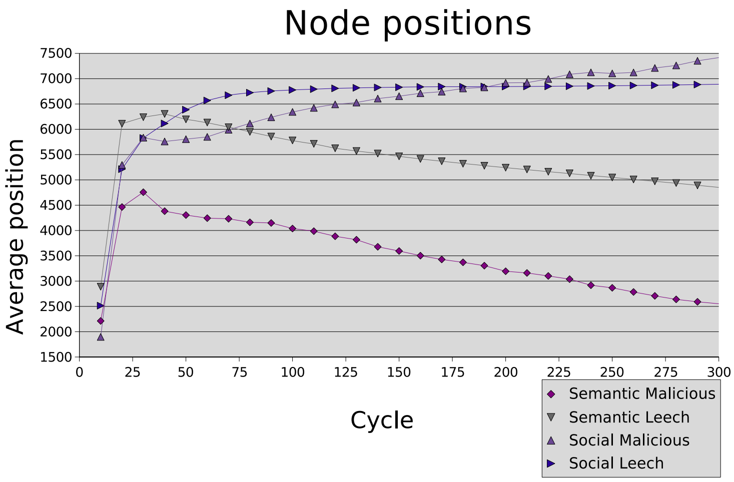


Fig. 4. Placement of nodes, lower is better. A high value of "average position" indicate that the node receive a bad rating.

Finally, we look at the ranking of nodes. Figure 4 shows the average rank of malicious and free-riders nodes in the system. These are only available for semantic and social routing, as random routing has no ranking of nodes. It is evident that semantic routing slowly incorporates the malicious and free-riders into the network. This is the expected behavior, as these nodes are regarded as relatively passive nodes. Notice that semantic routing rate malicious nodes as better than free-riders, due to their higher level of activity. The social routing ranks these nodes as the least interesting neighbors, as these nodes are not likely useful. Correctly, malicious nodes receive a worse rating than free-riders after only 200 cycles. By ranking malicious nodes and free-riders low, the social routing should provide an efficient search infrastructure even in the presence of such nodes.

6 Analysis

After the completion of the simulation, the topologies were kept for analysis. A topology was built for the FileList.org data trace by selecting the 25 semantically closest neighbors for each node. As all semantic routing is based on *Small World* networks, we analyzed both the simulated and the FileList.org topologies for the expected properties; grouping and separation level.

Node grouping is the presence of clustering of nodes based on semantic similarity. Grouping was measured by counting the number of common neighbors, the more common neighbors, the tighter the grouping. For every neighbor in a node's neighbor list, the lists were compared.

In figure 5 the number of common neighbors are shown. Grouping is very high in the FileList.org topology, with a peak at 19 common nodes. On average, the neighbor cache of two neighboring nodes are more than 50% equal (as only the 25 closest neighbors were calculated in the data trace).

The random topology is weakly grouped, which is to be expected. Semantic topology maps very closely to the data trace, with a very high level of grouping. The social topology is less grouped than the semantic topology. This is likely an effect of nodes with many resources being preferred over pure semantically close nodes due to node preference. Nodes with many resources likely have more diverse interests, filling their neighbor caches with nodes from several different interest groups. In other words, the “most popular” nodes in the network have more diverse social networks.

Both the semantic and social topologies has a second peak with extremely close nodes. This is likely nodes that have only very few interests, thus bonding very strongly. The FileList.org topology does not have a second peak as only 25 neighbors were calculated, while the simulations allowed 75 neighbors in order to build the network.

Finally, the level of separation of the topologies was calculated. The separation level is a measure of the distance between nodes. If a node directly connects to another node, they have separation 1. If an intermediate node must be used for the nodes to reach each other, they have a separation of 2. An analogy is that separation 1 is a “friend”, while separation 2 is a “friend of a friend”.

The separation level was calculated for each node by selecting 10 target nodes with at least one common resource with the node in question. The number of hops required to reach the target nodes was then found. This was done by first searching for the target in the node's own neighbor cache. If not present, the level was increased by one and the search extended. For the social routing, the top 10 neighbors were selected based on node preference. As semantic and random routing has no node preference, 10 random neighbors were selected. Due

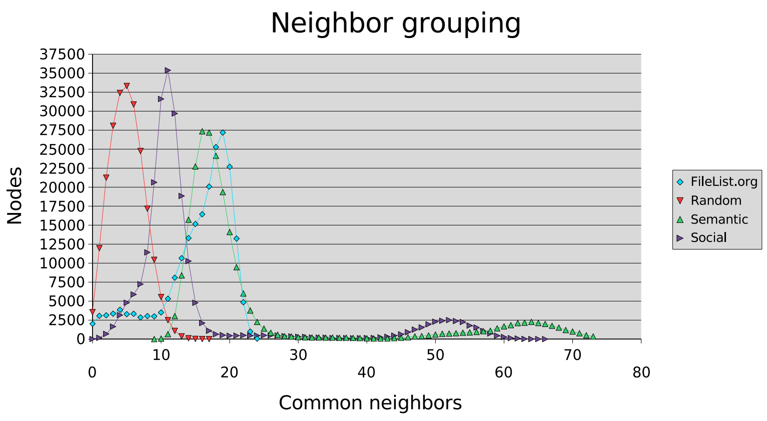


Fig. 5. Number of common nodes in neighbor caches

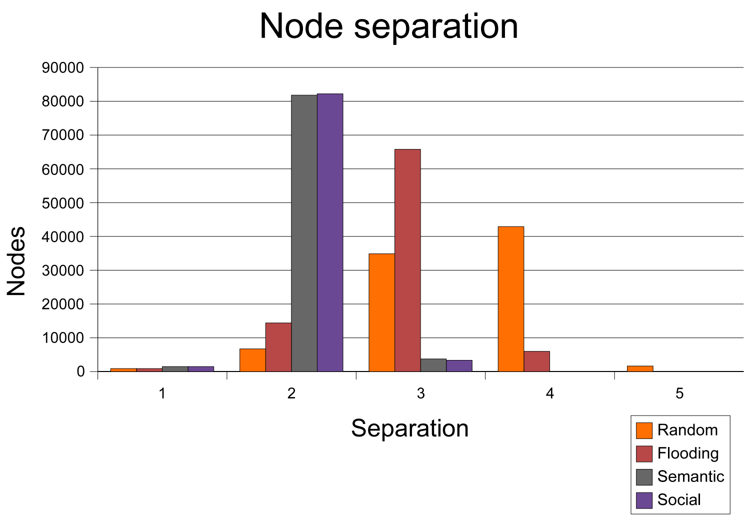


Fig. 6. Separation of the different topologies

to the bad performance of random routing, we also tried “flooding”, which used 25 random neighbors in stead of 10. If the target node was within the neighbor caches of any of the newly selected nodes, the search was successful and stopped. Otherwise, the search was extended again and new neighbors selected from each of the previously selected neighbors. This was repeated until either the node was found or until the set of nodes in the search became constant. In the last case, no route from the source to the target node was discovered.

Figure 6 shows the separation of the different topologies. Flooding clearly fares better than randomly routing to only 10 neighbors. Both random routing approaches found all target nodes. Semantic and social routing fared even better, with almost all target nodes only two jumps from the node itself. In the Filelist.org topology only about 3% of the target nodes was located, and was therefore not plotted. The Filelist.org topology is likely too "narrow minded", and is therefore comprised of many small islands. Allowing larger caches should at least partially remedy this, and adding some random nodes to the routing could assist in connecting the islands.

7 Concluding Remarks and Future Work

It is apparent that a very popular file sharing community, FileList.Org exhibit small world network properties. During simulations, both semantic and social routing fared well, with a short bootstrapping phase. Semantic and social routing are similar in most aspects, except that social routing rates malicious and free-riding nodes lower than semantic routing. This should make it possible to handle such nodes more gracefully. Note that social routing does not require a global agreement of what "malicious" means. *The Socialized.Net* allows the P2P overlay network to be directly influenced by the user's personal preference and background. Examples of personal preferences can be cultural differences (local "pop" music) or technical requirements (file formats or video sizes). For this paper, purely malicious nodes were used to make it easier to visualize how preference handle nodes with conflicting agendas or quality demands. Allowing all nodes to regard these nodes as malicious allows the analysis of their global rating to validate the effect.

During our simulations, a recall rate of approximately 55% was observed after 300 cycles of simulation both for semantic and social routing. This means that only half of the relevant resources are found. While this will likely improve as the network is left to run, it still indicates that *The Socialized.Net* is likely to miss some relevant resources during searches. However, the system is able to handle both malicious and free-riding nodes efficiently without any global rating. The efficiency, personalization and increased integrity might be of great value for applications that do not depend upon guaranteed recall rates. *The Socialized.Net* demonstrates that a fully decentralized social network can provide a powerful platform for personalized content navigation.

Future work should include large scale testing of the social search infrastructure. Also validation of our findings for data sets containing larger amounts of resources would be of great interest.

Acknowledgments

Thanks to the P2P group at Delft University, NL, specially Dr. Johan Pouwelse and Jelle Roozenburg for performing and providing the Filelist.Org data trace.

References

1. Knuth, D.E.: The Art of Computer Programming, vol. 3. Addison-Wesley, Reading (1973)
2. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web, Tech. rep., Stanford Digital Library Technologies Project (1998)
3. Oram, A. (ed.): Peer-to-peer: Harnessing the benefits of distruptive technologies, pp. 94–122. O'Reilly &Associates (2001)
4. Ripeanu, M., Foster, I., Iamnitchi, A.: Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. IEEE Internet Computing Journal 6(1)
5. Sigla, A., Rohrs, C.: Ultrapeers: Another step towards gnutella scalability, whitepaper, Lime Wire LLC
6. U. C. of Appeals for the Ninth Circuit, A&m records v napster, case number: 00-16401
7. Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Looking up data in p2p systems. Commun. ACM 46(2), 43–48 (2003)
8. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for internet applications. In: Proceedings of the 2001 ACM SIGCOMM Conference, pp. 149–160. ACM Press, New York (2001)
9. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric (2002)
10. Gupta, I., Birman, K., Linga, P., Demers, A., van Renesse, R.: Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, Springer, Heidelberg (2003)
11. Cohen, B.: Incentives build robustness in bittorrent
12. Pouwelse, J., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D., Reinders, M., van Steen, M., Sips, H.: Tribler: A social-based peer-to-peer system. In: Proceedings of the IPTPS 2006 (2006)
13. Joseph, S.: Neurogrid: Semantically routing queries in peer-to-peer networks. In: Proceedings of the International Workshop on Peer-to-Peer Computing, Pisa, Italy (2002)
14. Tempich, C., Staab, S., Wranik, A.: REMINDIN': Semantic query routing in peer-to-peer networks based on social metaphors. In: Proc. of the 13th World Wide Web Conference, pp. 640–649. ACM, New York, USA (2004)
15. Borch, N.T., Vognild, L.K.: Searching in variably connected p2p networks. In: Proc. of the Internation Conference on Pervasive computing and communications, Las Vegas, Nevada, US, pp. 806–812 (2004)
16. Borch, N.T.: Improving semantic routing efficiency. In: Proc. of the Hot P2P topics workshop, San Diego, US (2005)
17. Milgram, S.: The small world problem. Psychology Today 61 (1976)
18. Dodds, D.J.W.P.S., Muhamad, R.: An experimental study of search in global social networks. Science 8,301, 827–829 (2003)
19. Kleinberg, J.: The small-world phenomenon: An algorithmic perspective. In: Proceedings of the 32nd ACM Symposium on Theory of Computing, ACM Press, New York (2000)
20. Daniel Hughes, G.C., Walkerdine, J., Gibson, S.: Is deviant behavior the norm on p2p file-sharing networks?

Conflict Resolution of Boolean Operations by Integration in Real-Time Collaborative CAD Systems

Yang Zheng, Haifeng Shen, Steven Xia, and Chengzheng Sun

School of Computer Engineering, Nanyang Technological University,
Block N4, Nanyang Avenue, Singapore 639798
{zhen0042, ashfshen, stevenxia, czsun}@ntu.edu.sg

Abstract. Boolean operations are widely used in CAD applications to construct complex objects out of primitive ones. Conflict resolution of Boolean operations is a special and challenging issue in real-time collaborative CAD systems, which allow a group of geographically dispersed users to jointly perform design tasks over computer networks. In this paper, we contribute a novel conflict resolution technique that can retain the effects of individual conflicting Boolean operations by integrating them. This technique, named as CRIBO (Conflict Resolution by Integration for Boolean Operations), is in a sharp contrast to other ones that either desert the effects of some operations or keep the effects of different operations in different versions of the design. It is particularly good for collaborative CAD applications, where integration of different mindsets is a main source of creation and innovation. This technique lays a good foundation for resolving conflicting operations in design-oriented collaborative applications that require collective wisdom and stimulus of creation.

Keywords: CAD, Boolean operation, conflict resolution, real-time collaborative system, creative design.

1 Introduction

Computer Aided Design (CAD) is the use of a wide range of computer-based tools to assist engineers, architects and other design professionals in their design activities. In general, a CAD system is used to design, develop and optimize products, which can be goods used by end consumers or intermediate goods used in other products. It is also extensively used in the design of tools and machinery used in the manufacturing of components as well as in the drafting and design of all types of buildings, from small residential types (e.g., houses) to large commercial and industrial structures (e.g., hospitals and factories).

Collaboration has been increasingly needed in the CAD community since early 1990s. As design tasks are getting more and more complex, designers have to work together more and more often as a team rather than individually. With the help of computer networks and online collaboration tools, it is possible to effectively fill up communication gaps among geographically dispersed designers and to significantly reduce the time in both the design and the implementation phase in a product development cycle. CAD designers/draftsmen may use a real-time collaborative CAD system to clarify doubts quickly without leaving their desktops. Designers and the project leader can track bugs and discuss changes in front of computers instead of face-to-face. Engineers from different streams or departments can collaborate to work around a conflicting situation. Furthermore, a collaborative CAD system can be used to interactively demonstrate its main features to customers, or to train or get feedback from customers.

Before the advent of real-time collaborative CAD systems, collaboration is mainly facilitated by various online chatting applications such as ICQ [2] and MSN [10]. However, this kind of collaboration could hardly meet the specific needs of technical discussions, in which verbal communication is not enough to express designers' ideas clearly and to represent graphical features accurately. Some designers use application-sharing systems such as Microsoft NetMeeting [11] to share their CAD applications for real-time collaboration. These systems, however, do not support concurrent work and are not responsive in Wide Area Networking (WAN) environments.

To meet specific requirements in the CAD community, a few real-time collaborative CAD systems have been developed, which allow designers to manipulate the same design documents at the same time. To achieve high responsiveness in a networking environment, shared documents are usually replicated at each collaborating site so that editing operations can be performed at local sites immediately and then propagated to remote sites.

Consistency maintenance is a fundamental issue in these real-time collaborative CAD systems, especially in the presence of conflicting operations. In a distributed environment, users may concurrently issue operations that semantically or syntactically conflict with each other. For example, two "*move*" operations may conflict with each other when two users concurrently try to move a particular object to different positions.

Several strategies have been proposed to solve conflicting operations. One of those strategies is to adopt a conflict prevention strategy based on pessimistic concurrency control mechanisms such as locking and turn-taking. In these systems, a user has to gain the "ownership" of the target objects before she/he can edit it, thus preventing other users from generating conflicting operations on the same object. For example, the TOBACO system [8] adopts a floor control mechanism; the Cooperative ARCADE system [13] uses a locking mechanism. This means that at any moment in time, only one active user (i.e., the one who holds the floor/lock on the particular objects) is allowed to perform her/his design. This could significantly degrade the system's responsiveness and furthermore make a design process tedious.

Another alternative conflict resolution approach is by means of serialization, which ensures that the effect of executing a group of concurrent operations be the same as if these operations were executed in the same order at all sites. If there is any conflict among concurrent operations, only the effect of the last operation (based on the total ordering) will be kept. This approach was mostly used in early collaborative systems such as GroupDesign [4] and LICRA [3]. However, as a design process is usually complex, it is undesirable to keep the effect of only one operation while destroying the effects of others.

A more advanced conflict resolution technique is Multi-Versioning (MV) [9, 14]. The MV technique preserves all users' work by keeping multiple versions of shared artifacts. Such a strategy provides better feedback to the users, helps the users to better understand the nature of the conflicts, and to better adjust and coordinate their actions in the face of conflicts. However, MV is not suitable for design-oriented collaborative systems because keeping conflicting operations in different versions of shared objects does not help designers analyze conflicts as a whole or make use of conflicts to stimulate creative design.

In this paper, we contribute a novel technique to resolve conflicting Boolean operations. The technique, named as CRIBO (Conflict Resolution by Integration for Boolean Operations), is able to retain the effects of individual conflicting Boolean operations and integrate them as a whole. Compared with previous conflict resolution techniques that either desert the effects of some conflicting operations or keep the effects of different operations in different versions of the design, CRIBO provides designers with a more comprehensive and straightforward view of the conflict. This is particularly good for design-oriented applications as integration of different mindsets is a main source of creation and innovation.

The rest of this paper is organized as follows. In Section 2, conflicting Boolean operations are introduced and their special characteristics are analyzed. Section 3 then explains how to resolve conflicting Boolean operations using CRIBO in detail. This includes selecting proper techniques to support CRIBO and specifying rules on how to determine the results for a group of conflicting Boolean operations. In Section 4, we discuss how to use CRIBO to better support collaborative design scenarios that involve Boolean operations and other typical CAD operations. Finally, major contributions of this paper and the future work are summarized in Section 5.

2 Conflicting Boolean Operations

Consistency maintenance in the presence of conflicting operations is a challenging issue in collaborative systems. Conflicts are domain-specific and application-specific. Our investigation reveals that operations in the CAD domain have several special characteristics that raise challenging but interesting issues in the face of conflicts. For example, the process of performing a particular CAD operation takes relatively longer time, as users have to interact with the system to specify parameters and select options. In addition, a CAD operation may involve a considerable number of objects, which significantly increases the possibility that concurrent operations have common target objects and may consequently lead to potential conflicts.

This paper takes Boolean operations, a representative category of CAD operations, as an example to illustrate the characteristics of conflicting CAD operations and to devise an effective conflict resolution technique for CAD operations.

2.1 Boolean Operations

Nowadays, most off-the-shelf CAD systems (e.g., AutoCAD [1] and OneSpace Modeling [12]) are object-based, in which geometrical objects such as rectangles, circles, polygons and 3D solids can be created and modified. In these systems, complex artificial objects can be constructed out of primitive ones using Boolean operations. In this paper, Boolean operations refer to the three primitive ones *Union*, *Subtract* and *Intersect*, which are supported by most CAD systems. The *Union* operation, denoted as "∪", is used to join two or more objects into one based on the total geometry of all. The *Subtract* operation, denoted as "–", is used to subtract one or more objects from another to create an object based on the remaining geometry. The *Intersect* operation, denoted as "∩", is used to create a single object from two or more objects based on the intersected geometry. For the good of presentation, we express a Boolean operation with its target objects and its operator. That is, operation $O = A \cup B$ is to *Union* two objects A and B; operation $O = A - B$ is to *Subtract* object B from A; and operation $O = A \cap B$ is to *Intersect* two objects A and B. Figure 1 illustrates the three primitive Boolean operations and their effects[1].

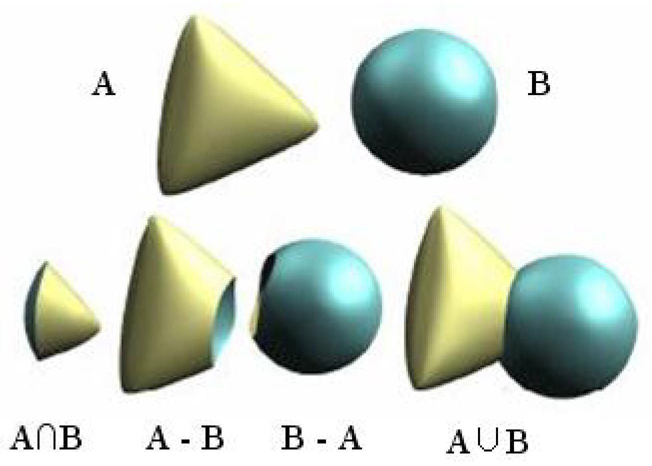
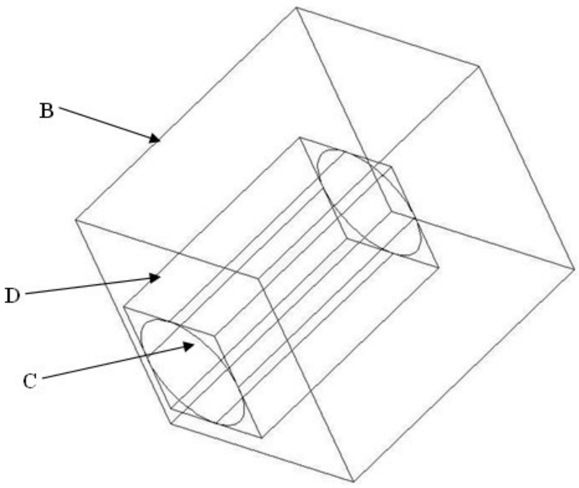


Fig. 1. Primitive Boolean operations and their effects

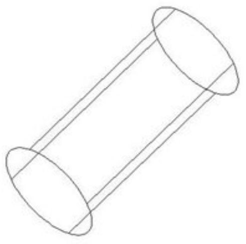
2.2 Conflicting Boolean Operations

A Boolean operation has several target objects. In a real-time collaborative CAD environment, when different designers concurrently issue their own Boolean operations, it is possible that those Boolean operations have common target objects. To illustrate this, let's look at a collaborative design scenario. Assume two designers are working together to mold a cubical diamond (marked as *D* in Figure 2 (a)). To get a desirable shape, a cylinder (marked as *C* in Figure 2 (a)) and a box (marked as *B* in Figure 2 (a)) are created. All the three objects are 3D solids. Suppose the two

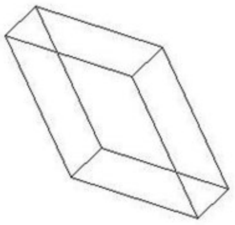
[1] It should be pointed out that there is a specific geometrical relationship between objects *A* and *B* and they are actually overlapping with each other. However, this geometrical relationship is not revealed explicitly in Figure1. This is done on purpose so that readers can get a clear idea about the effects of Boolean operations.



(a) Three 3D objects in a diamond design scenario



(b) The geometrical effect of $D \cap C$



(c) The geometrical effect of $D–B$

Fig. 2. Conflicting Boolean operations and their individual effects

designers concurrently issue Boolean operations that target at D, C and B. The first designer's operation is $O_1 = D \cap C$, whereas the second designer's operation is $O_2 = D - B$.

Definition 1 Conflicting Boolean operations "$\otimes$". Two Boolean operations O_1 and O_2[2] *conflict with each other, denoted as "$O_1 \otimes O_2$", if and only if $O_1 \parallel O_2$ (i.e., they are concurrent)*[3] *and Target (O_1) $\cap$ Target (O_2) $\neq$ null; or there exists another concurrent operation O_x, such that $O_1 \otimes O_x$ and $O_x \otimes O_2$. Here, Target (O) denotes the set of objects targeted by a Boolean operation O.*

[2] In this paper, we suppose O_1 and O_2 are always generated at the same document state.

[3] Two operations are concurrent if they are generated without any knowledge of each other. For the formal definition of concurrent operations, readers may refer to [6, 14].

For example, in Figure 2, $O_1 \otimes O_2$ because $O_1 \parallel O_2$ and Target (O_1) $\cap$ Target (O_2) = $\{D\} \neq null$.

3 Resolving Conflicting Boolean Operations by Integration

3.1 Resolving Conflicting Boolean Operations

In the face of conflicting Boolean operations, it is important to preserve operations' intentions using appropriate techniques to resolve the conflicts. This implies several requirements. First, after resolving the conflicts, only one version that has the effects of all the operations should be produced. This allows designers to analyze the conflicts as a whole and to make use of the conflicts to stimulate design innovation. Second, for an arbitrary group of conflicting Boolean operations, rules should exist to precisely determine the effects of the operations. These rules should be independent of the execution order of those Boolean operations so that consistency can be maintained [14]. Third, the produced effects should be understandable to users so that they can assess the situation, react accordingly and make use of the effects in a collaborative design environment.

We propose an innovative conflict resolution technique, which can retain the effects of individual conflicting Boolean operations by integrating them. The biggest challenge here is how to achieve the integration. To explain our solution, we will first introduce some notations. First, given a Boolean operation *O*, its effect, denoted as *E (O)*, is the geometrical effect when *O* is successfully performed. Second, given a group of conflicting Boolean operations $O_1, O_2, ..., O_n$, their Integrated Effect is denoted as *IE* ($O_1 \oplus O_2 \oplus ... \oplus O_n$), where operator "$\oplus$" is *union*, *subtract* or *intersect*. In a collaborative environment, *IE* ($O_1 \oplus O_2 \oplus ... \oplus O_n$) defines the effect after those operations are successfully performed. Third, given a Boolean operation expression whose targeting objects are $A_1, A_2, ..., A_m$, its effect is denoted as $[\![A_1 \oplus A_2 \oplus \ldots \oplus A_m]\!]$, where operator "$\oplus$" is *union*, *subtract* or *intersect*. It should be pointed out that the definitions of *E* and *IE* are defined at the *operation* level whereas "$[\![\ ]\!]$" is defined at the *object* level. Therefore, from users' point of view, an *E* or *IE* expression can always be mapped to an object expression with the format of $[\![A_1 \oplus A_2 \oplus \ldots \oplus A_m]\!]$, which provides a more straightforward way to understand the effects of an *E* or *IE* expression.

We have investigated possible techniques to integrate a group of conflicting Boolean operations. The results revealed that the most appropriate one is to use *Union* to integrate those operations. For the example in Figure 2, after integrating O_1 and O_2 using *Union*, the result is $[\![(D \cap C) \cup (D - B)]\!]$ (as shown in Figure 3).

The advantages of selecting *Union* to resolve conflicting Boolean operations are as follows. First, a single version can be obtained from the *N* individual versions that are created by a group of conflicting Boolean operations $O_1, O_2, ..., O_n$. Based on the semantics of *Union*, all individual operations' effects (i.e., *E* (O_1), *E* (O_2), ..., *E* (O_n))

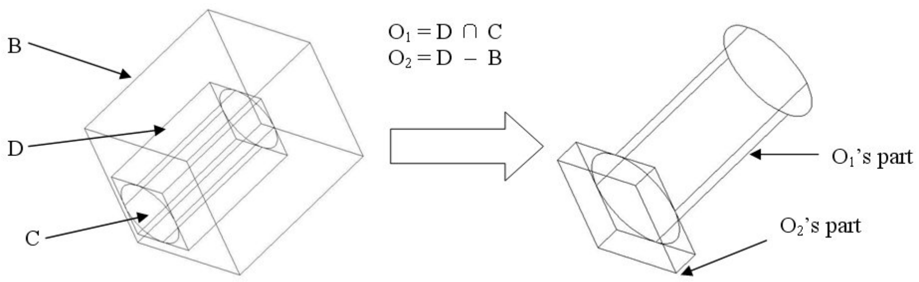


Fig. 3. The obtained effect after integrating O_1 and O_2

will be interpreted in the produced version and reflected to designers. It should be pointed out that, on some occasions, the effects of the produced version could also be achieved through a serializable execution (to be explained in Section 3.3). However, compared with the serialized one in which the resultant effects can be expected, our approach produces totally unexpected effects, which could well be the source of creative design (to be explained in Section 4).

Second, by using *Union*, the produced version will be consistent at different sites. This is because the *Union* operator conforms to the *Commutative Law (i.e.,* $O_1 \cup O_2 = O_2 \cup O_1$*)* and the *Associative Law (i.e.,* $(O_1 \cup O_2) \cup O_3 = O_1 \cup (O_2 \cup O_3)$*)*. Therefore, given an arbitrary group of conflicting Boolean operations $O_1, O_2, ..., O_n$, no matter in which order the operations are executed at a particular site, the obtained geometrical effect is always the same as $IE\ (O_1 \cup O_2 ... \cup O_n)$.

Third, as *Union* is an operator that belongs to the CAD community, it is relatively easy for users to understand the produced result as it is natural for them to analyze it using their knowledge about the operator.

3.2 Rules

According to our solution, given a group of conflicting Boolean operations $O_1, O_2, \ldots, O_n$, the effect of the produced version is $IE\ (O_1 \cup O_2, \ldots \cup O_n)$. As mentioned in the previous section, *Union* conforms to both the *Commutative Law* and the *Associative Law*. As a result, it can be drawn that $O_1 \cup O_2 \ldots \cup O_n = (O_1 \cup O_2, \ldots \cup O_{n-1}) \cup O_n$, which means that the effect of *Unioning N* objects is equal to that of *Unioning* one object with the other *N - 1* objects. Assuming that there are two operations O_i and O_j, where $1 <= i < j <= n$, if we can correctly determine $IE\ (O_i \cup O_j)$, then we can correctly determine $IE\ (O_1 \cup O_2 \ldots \cup O_n)$. Therefore, we only need to focus on how to draw rules to determine $IE\ (O_i \cup O_j)$.

We first start from the simplest scenario, in which each of the two conflicting Boolean operations targets at only two objects and one of the two objects is targeted by both operations, e.g., *Target* (O_1) = *{A, B}* and *Target* (O_2) = *{B, C}*, and *Target* $(O_1) \cap$ *Target* (O_2) = *{B}*. All possible results of $IE\ (O_1 \cup O_2)$ are listed in Table 1.

Table 1. The effects of resolving conflicting Boolean operations

O_2 / O_1	**B∩C**	**B∪C**	**B - C**	**C - B**
A∩B	B∩(A∪C)	B∪C	(A∩B)∪(B - C)	(A∩B)∪(C - B)
A∪B	A∪B	A∪B∪C	A∪B	A∪B∪C
A - B	(A - B)∪(B∩C)	A∪B∪C	(A - B)∪(B - C)	(A□C) - B
B - A	(B - A)∪(B∩C)	B∪C	(B - A)∪(B - C)	(B - A)∪(C - B)

The rules between two arbitrary conflicting Boolean operations start from the above table and in the end four rules are drawn. In the following part of this section, we will first introduce the four rules and then make an analysis on them.

For the good of presentation, we first define complex object sets *A*, *B*, *C*, *D* and *E* as operands in conflicting Boolean operations. These object sets are

$$A = A_1 \bigcup ... \bigcup A_K,\ B = B_1 \bigcup ... \bigcup B_L,\ C = C_1 \bigcup ... \bigcup C_M,$$

$$D = D_1 \bigcup ... \bigcup D_N, \text{ and } E = E_1 \bigcup ... \bigcup E_P$$

where A_k ($1 \le k \le K$), B_l ($1 \le l \le L$), C_m ($1 \le m \le M$), D_n ($1 \le n \le N$) and E_p ($1 \le p \le P$) are primitive objects that are the smallest units and not required to be further split in the given Boolean operations.

For Boolean operators *Union* ($\bigcup$), *Intersect* ($\bigcap$), *Subtract* ($-$), we categorize *Union* and *Intersect* as *positive* operators since these two operators conform to the *Commutative Law*, whereas *Subtract* is considered as *negative* operator as it dose not conform to the *Commutative Law*.

Given two conflicting Boolean operations O_i and O_j, where $O_i = A \bigcup B$ and $O_j = B \bigcup C$, based on Table 1, the effect of integrating O_i and O_j is $[\![A \bigcup B \bigcup C]\!]$. If the two conflicting Boolean operations are O_i and O_j, where $O_i = A \bigcap B$ and $O_j = B \bigcap C$, the effect of integrating O_i and O_j is $[\![(A \bigcup C) \bigcap B]\!]$. Based on these two results, we can derive Rule 1 as follows.

Rule 1. Conflicting Boolean operations with the same positive operators. Given two conflicting Boolean operations O_i and O_j, where operator "$\circ$" is either Union or Intersect, if $O_i = A \circ B$ and $O_j = B \circ C$, then the effect of integrating O_i and O_j is $[\![B \circ (A \bigcup C)]\!]$.

Rule 2. Conflicting Boolean operations with different positive operators. Given two conflicting Boolean operations O_i and O_j, where $O_i = A \bigcup B$ and $O_j = B \bigcap C$, the effect of integrating O_i and O_j is $[\![A \bigcup B]\!]$, which is the same as E (O_i).

Rule 3. Conflicting Boolean Operations with Union and Subtract. Given two conflicting Boolean operations O_i and O_j, where $O_i = A \bigcup B$, $O_j = C \bigcup B_{S1} - D \bigcup B_{S2}$ ($B_{S1} \bigcap B_{S2} = null, B_{S1} \bigcup B_{S2} = B$)[4], the effect of integrating O_i and O_j is $\llbracket A \bigcup B \bigcup (C - D) \rrbracket$.

Rule 4. Conflicting Boolean Operations with Intersect and Subtract. Given two conflicting Boolean operations O_i and O_j, where $O_i = D \bigcup C_{X1} - E \bigcup C_{X2}$ ($C_{X1} \bigcap C_{X2} = null, C_{X1} \bigcup C_{X2} = C$), and $O_j = A \bigcap B$ or $O_j = A \bigcup C_{Y1} - B \bigcup C_{Y2}$ ($C_{Y1} \bigcap C_{Y2} = null, C_{Y1} \bigcup C_{Y2} = C$), then the effect of integrating O_i and O_j is $E(O_i) \bigcup E(O_j)$.

3.3 Rules Analysis

In the face of an arbitrary group of conflicting Boolean operations, the four rules can preserve individual operations' effects and integrate them as a whole. From designers' perspective, the integrated effect is not a premeditated one and thus is new to them. In a collaborative design environment, this new effect could be a great stimulus to design innovation. That is, in a design process, when conflicting Boolean operations are generated, it is usually because there is no obvious or pre-defined solution to those commonly targeted objects (Otherwise, the designers would not target at the same objects and instead, would perform operations on their "own" objects). In this case, it is vital that a designer has a comprehensive and yet easy-to-understand way to learn other designers' ideas, which is provided by applying CRIBO. As a result, a designer can easily analyze other designers' ideas and thus achieve design efficiency.

It should be pointed out that the integrated effect may also be achieved in a single-user design environment. To *Rule 1*, *Rule 2* and the *Rule 3*, a target object appears once and only once in the rule expression, which means that the resulted effect could be achieved by sequentially performing a set of Boolean operations on the target objects. Nevertheless, efficiency can be significantly increased by achieving parallel computing in a collaborative design environment as individual operations are performed at different sites.

4 Significance of the CRIBO Technique in Supporting Creative Design

The conflict resolution technique CRIBO distinguishes itself from other techniques in that it can retain the effects of individual conflicting operations by integrating them. This characteristic could significantly benefit a design process, where integration of different mindsets is a main source of creation and innovation. In this section, we

[4] i.e., B_{s1} and B_{s2} are disjoint subsets of B, and all the elements of B belong to either B_{s1} or B_{s2} but not both.

provide concrete examples to show how CRIBO is applied to support collaborative designs that involve Boolean operations and other typical CAD operations.

4.1 Obtain "Hard-to-Achieve" Effects

As mentioned in previous sections, Boolean operations have been widely used in design representation, analysis and manufacturing. In these domains, the inherent complexities of geometrical configuration require strong capability from the CAD tools for the purpose of complex modeling. These applications, with specific requirements in their respective domains [5, 15], require comprehensive and complex modeling functionalities. As a result, the three basic Boolean operations are not enough, and more complex and domain-specific Boolean operations are required.

Our investigation on the complex Boolean operations, however, reveals that those operations are all based on the three basic Boolean operations and therefore can be achieved with a proper combination of the basic Boolean operations. However, it is non-trivial to implement complex Boolean operations by combining basic Boolean operations in single-user design environment.

For example, the *Combine* operation is a widely-used complex Boolean operation [16]. As shown in Figure 4, the *Combine* operation is like the *Union* operation but removes the portions in common (e.g., the geometrically overlapping part), i.e., *Combine (A, B) = (A – B) U (B – A)*. It is obvious that the *Combine* operation cannot be achieved by sequentially executing Boolean operations as it requires a pair of *A* and *B*. For example, to achieve this operation in AutoCAD, designers have to make a copy of both A and B, locate them at exactly the same position, perform *A – B* and *B – A* respectively, and in the end *union* them together.

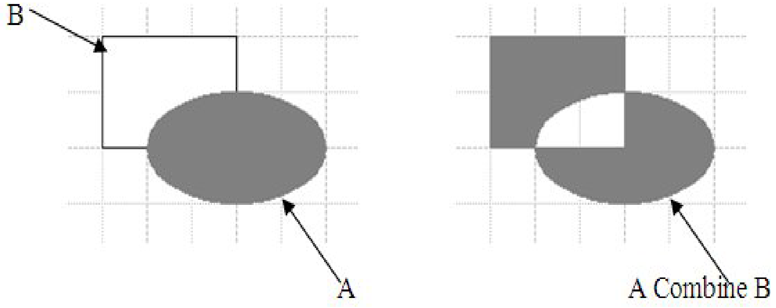


Fig. 4. The *Combine* operation

Such a process is tedious and undesirable in a design session. However, this could be easily solved in a collaborative design environment by purposely generating conflicting Boolean operations and using our conflict resolution strategy to achieve the desired effects. For example, given two users in a real-time collaborative design session, one user performs *A – B* and the other performs *B – A* concurrently. Under the well-defined rules of our conflict resolution strategy (actually based on *Rule 4*), we can immediately get the geometry of the required result (i.e., $(A - B) \cup (B - A)$), which significantly saves design efforts and simplifies the design process.

4.2 Stimulate Design Innovation

Another significant contribution of our conflict resolution strategy is that it can not only resolve conflicts, but also provide new effects based on the conflicting operations. We believe that these effects could dramatically stimulate design innovation and in the long run could significantly benefit the design of collaborative systems.

To the best of our knowledge, previous conflict resolution strategies in the CAD community consider a design process as purely a technical process, in which conflicts are usually regarded as being abnormal and should be avoided as soon as possible. However, a collaborative design process is not as simple as this and other factors, especially social factors between different designers (who are usually from different domains) should also be taken into account. As argued in [7], a collaborative design process is mostly a socio-technical process, in which conflicts must be systematically and explicitly dealt with as a resource to drive design innovations.

For example, consider a collaborative design scenario in Figure 5. Three users are trying to get creative shapes based on the three primitive ones, which are represented as A, B and C. At a particular time, they concurrently issue their own Boolean operations as $O_1 = A \cap B$, $O_2 = A \cap C$ and $O_3 = B \cap C$. Three effects are achieved as $[\![A \cap B]\!]$, $[\![A \cap C]\!]$ and $[\![B \cap C]\!]$, but none has obvious meanings. However, these operations conflict with each other and as a result, the underlying collaborative system *unions* the three effects into a new one, which resembles a simple model of an airscrew.

This example demonstrates that a collaborative CAD system can not only resolve conflicts, but also achieve new effects based on conflicting operations. More importantly, these new effects are not randomly generated, but are based on well-defined rules. This ensures that the effects have relationships with all the individual effects and thus could very likely stimulate designers' ideas.

4.3 Applying CRIBO to Other CAD Operations

As proved in previous sections, CRIBO can resolve conflicting Boolean operations by integration and provide new effects to stimulate design innovation. Our investigation shows that this technique lays a good foundation for resolving other conflicting operations in design-oriented collaborative applications. The key to achieving this, however, is to select the most appropriate approach to integrate those conflicting operations. Such a selection is non-trivial because the integration depends on several aspects, such as the conflicting operations' semantics and their working environments. In the following section, we will explain how to find out the appropriate approaches to resolving conflicts that involve other typical CAD operations, including the *rotate* operation and the *color* operation.

4.3.1 Apply CRIBO to the *Rotate* Operation

Rotate is a typical CAD operation, which allows users to rotate an object around a specified point based on a particular angle value. In a real-time collaborative design environment, two *rotate* operations may conflict with each other if they concurrently target at the same object but attempt to rotate it in different directions.

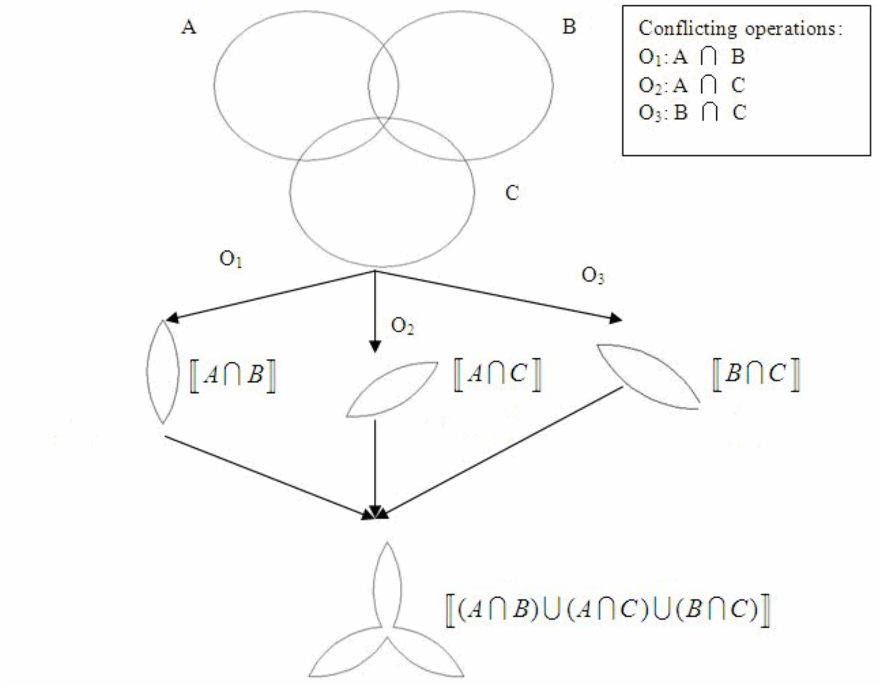


Fig. 5. Achieve the effects of an airscrew by resolving conflicts

Let's look at a simple but interesting scenario. Suppose two designers are collaborating to design the layout of a living room. At a time, they are to determine where to arrange the bed, as shown in Figure 6 (a). Both designers have their own arrangement criteria. The first designer wants the bed to be near the window so that it can gain more sunshine whereas the second designer wants it to be opposite to the door to avoid noise at the corridor. As a result, they concurrently issue their own *rotate* operations and locate the bed according to their preferences, as shown in Figure 6 (b) and (c), respectively.

Obviously, the two designers' intentions can coexist (as they are based on different criteria) and therefore should both be reflected in the design. However, previous collaborative design systems would consider them as a conflict as they rotate the same objects to different positions and directions. As a result, designers may receive messages about the conflicts, communicate to explain their intentions to each other and then try to find another solution to satisfy both criteria.

This tedious process could be effectively shortened by using CRIBO. For example, suppose a *rotate* operation is represented as *Rotate (object, base-point, rotate-angle).*

In the face of two conflicting *rotate* operations, we can integrate both operations' effects into one, in which the centroid of the *object* is the mid-point of the two *base-points* and the new direction is an average of the *rotate-angles*. Based on this rule, we can determine the arrangement of the bed by integrating both designers' operations. As shown in Figure 6 (d), the bed is located at a position that is opposite to the door and not far from the window, which could satisfy both designers' intentions.

4.3.2 Apply CRIBO to *Color* Operation

Color is a vastly-used CAD operation. It can fill a graphic object based on user specified colors or texture patterns. In a real-time collaborative design environment, two *color* operations may conflict with each other if they concurrently target at the same object but fills it with different colors.

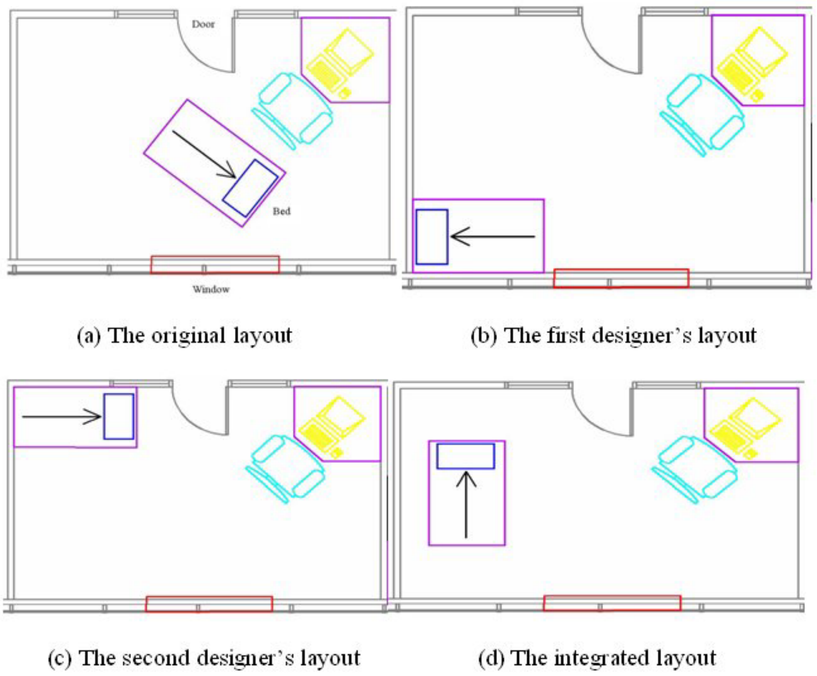

Door
Bed
Window
(a) The original layout
(b) The first designer's layout
(c) The second designer's layout
(d) The integrated layout

Fig. 6. Resolve conflicting *rotate* operations using CRIBO

To explicitly explain this scenario, let's look at an example. Suppose three designers are in a real-time design process to determine which colors should be filled in to an object *A*, whose original color is *black*. For the good of presentation, we assume a *color* operation is represented as *Color (object, old-color, new-color)*. Suppose the designers concurrently issue three *color* operations O_1, O_2 and O_3. These operations are O_1 = *color (A, black, red)*, O_2 = *color (A, black, green)* and O_3 = *color (A, black, blue)*. Obviously, the three operations conflict with each other as they are trying to change the same object (i.e., *A*) to different colors.

This conflict can be resolved using CRIBO by integrating the three colors into another based on classical color storage scheme in graphic design. That is, for three operations O_1, O_2 and O_3, suppose O_1's new color is represented as an RGB triple $<R_1, G_1, B_1>$, O_2's is represented as $<R_2, G_2, B_2>$ and O_3's is represented as $<R_3, G_3, B_3>$. Then the mixture of O_1, O_2 and O_3's new colors is equal to $<(R_1+R_2+R_3) \bmod 256, (G_1+G_2+G_3) \bmod 256, (B_1+B_2+B_3) \bmod 256>$. Based on this rule, the integration of O_1, O_2 and O_3 in this scenario is filling the object *A* with the color of *white*, which is "new" to all the three designers.

Furthermore, the *white* color is not the only possible "new" color achievable using CRIBO. As shown in Table 2, we can altogether obtain eight colors from the three original colors. To the best of our knowledge, existing conflict resolution strategies can at most achieve the first four color effects (i.e., *black*, *red*, *green* and *blue*). However, CRIBO can obtain not only these four colors but also four more colors (i.e., *yellow*, *purple*, *cyan* and *white*) that could not directly be achieved by using any other strategies. We believe that these new effects could greatly provide designers with collective wisdom and stimulus of creation, especially in those complex design scenarios.

It should be pointed out that the aim of using CRIBO is to provide designers with several options to resolve a conflict rather than to produce a specific result. By using CRIBO, designers are able to have all the possible results (e.g., the eight resultant colors in Table 2). After negotiation and discussion, they can inform the underlying collaborative design systems about the selected effects. Such an approach ensures that designers' ideas are correctly realized in the design process and prevent potential disputes and conflicts from happening

Table 2. Differnt color effects achieved by integration

Color	Integrated operations
1. Black	(no operation integrated)
2. Red	(O_1)
3. Green	(O_2)
4. Blue	(O_3)
5. Yellow	(O_1, O_2)
6. Purple	(O_1, O_3)
7. Cyan	(O_2, O_3)
8. White	(O_1, O_2, O_3)

5 Conclusion

Real-time collaborative CAD systems allow a group of users to view and edit the same design document at the same time from geographically dispersed sites connected by networks. With several members working on a project collaboratively and concurrently, it is possible to shorten design cycle, improve design quality and reduce design cost. As those collaborative CAD systems adopt a replicated architecture to achieve quick local responsiveness, consistency maintenance is a fundamental issue, especially in the face of conflicting operations.

This article contributes a novel technique CRIBO to resolve conflicting Boolean operations. CRIBO distinguishes itself from previous ones in that it can retain the effects of all the conflicting Boolean operations by integration. Our research shows that such an approach could provide as much information as possible for users to understand the conflicts. Additionally, the technique can be used to create new effects that could significantly increase design efficiency and stimulate design innovation. The paper also explicitly specifies how to obtain the desired effects in the face of a group of conflicting Boolean operations.

Future work of this paper includes several aspects. The CRIBO technique is being implemented in a collaborative AutoCAD system. We are moving towards making the system publicly demonstrable. We plan to apply the technique to resolving other conflicting operations in CAD applications and other domains such as graphic systems and Word processing tools.

References

1. Autodesk Inc.: AutoCAD: AutoCAD products information, http://usa.autodesk.com/adsk/servlet/index?siteID=123112
2. ICQ Inc.: ICQ Community people search and messaging service, http://www.icq.com
3. Kanawati, R.: LICRA: A replicated-data management algorithm for distributed synchronous groupware applications. Parallel Computing 22(13), 1733–1746 (1997)
4. Karsenty, A., Tronche, C., Beaudouin-Lafon, M.: Groupdesign: Shared editing in a heterogeneous environment. Usenix Journal of Computing Systems 6(2), 167–195 (1993)
5. Kumar, V., Dutta, D.: An approach to modeling and representation of heterogeneous objects. Journal of Mechanical Design 120(4), 659–667 (1998)
6. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7), 558–565 (1978)
7. Lu, S., Cai, J., Burkett, W., Udwadia, F.: A methodology for collaborative design process and conflict analysis. CIRP Annals - Manufacturing Technology 49(1), 69–73 (2000)
8. Lukas, U.: Collaborative geometric modeling using CORBA services. In: Proceedings of the ECSCW 1997 Workshop on Object-Oriented GroupWare Platforms OOGP 1997, Lancaster, UK, pp. 91–92 (1997)
9. Moran, T., McCall, K., Van Melle, B., Pedersen, E., Halasz, F.: Some design principles for sharing in Tivoli, a whiteboard meeting-support tool. In: Greenberg, S. (ed.) Groupware for Real-time Drawings: A Designer's Guide, pp. 24–36. McGraw-Hill International, UK (1995)
10. Microsoft Corporation, M.S.N.: Windows Live Messenger, http://im.live.com/messenger/im/home/?source=MSNTDLINK
11. Microsoft Corporation. NetMeeting Home, http://www.microsoft.com/windows/netmeeting/
12. CoCreate Inc. OneSpace Modeling.: CoCreate OneSpace Modeling, http://www.cocreate.com/designer_modeling.cfm
13. Stork, A., Jasonch, U.: A collaborative Engineering Environment. In: Proceedings of TeamCAD 1997 Workshop on Collaborative Design, Atlanta, USA, pp. 25–33 (1997)
14. Sun, C., Chen, D.: Consistency maintenance in real-time collaborative graphics editing systems. ACM Transactions on Computer-Human Interaction 9(1), 1–41 (2002)
15. Sun, W., Lin, F., Hu, X.: Computer-aided design and modeling of composite unit cells. Composite Science and Technology 61, 289–299 (2001)
16. Microsoft Corporation. Visio Home – Microsoft Office Online, http://office.microsoft.com/en-us/visio/default.aspx

Trust Extension Device: Providing Mobility and Portability of Trust in Cooperative Information Systems

Surya Nepal, John Zic, Hon Hwang, and David Moreland

ICT Centre PO Box 76 Epping NSW 1710 Australia
{Surya.Nepal, John.Zic, Hon.Hwang, David.Moreland}@csiro.au

Abstract. One method for establishing a trust relationship between a server and its clients in a co-operative information system is to use a digital certificate. The use of digital certificates bound to a particular machine works well under the assumption that the underlying computing and networking infrastructure is managed by a single enterprise. Furthermore, managed infrastructures are assumed to have a controlled operational environment, including execution of a standard set of applications and operating system. These assumptions are also valid for recent proposals on establishing trust using hardware-supported systems based on a Trusted Computing Module (TPM) cryptographic microcontroller. However, these assumptions do not hold in today's cooperative information systems. Clients are mobile and work using network connections that go beyond the administrative boundaries of the enterprise. In this paper, we propose a novel technology, called Trust Extension Device (TED), which enables mobility and portability of trust in cooperative information systems that works in a heterogeneous environment. The paper provides an overview of the technology by describing its design, a conceptual implementation and its use in an application scenario.

1 Introduction

Traditional cooperative information systems enable interactions between clients and servers within a controlled computing and networking infrastructure belonging to a single enterprise. However, it is now possible to develop such systems that go beyond a single enterprise's administrative domain due to the availability of new architectural approaches and software technologies (Web Services and Service Oriented Architecture), networking (Internet) and mobile computing devices (e.g. laptops, PDAs). As a result, we have seen a growing number of cooperative information systems developed to run in heterogeneous, open and hostile environments. Though such systems provide greater flexibility, they present a new set of challenges on three critical issues of information systems: trust, security and privacy. Though these issues are equally important, and related to each other, our focus in this paper is primarily on *trust-enhanced security*.

In emerging internet-scale cooperative information systems, agents (or clients) on behalf of their enterprises may connect to the enterprise system to access information using a range of devices from personal digital assistant (PDA) to desktop computers operating under a variety of platform configurations via the Internet. It is important to ensure that the agents are genuine before the enterprise releases information held at its controlled, managed environments. Without having a proper method in place for establishing trust, some attackers may spoof "real" agents and perform certain actions, that fool enterprise systems into believing that genuine agents/clients have performed these actions. Our aim in this paper is to define a way of establishing trust between enterprises and their agents that operate in an untrusted, hostile and heterogeneous environment.

What is *trust*? Trust has been defined variously in a range of literature, from philosophy to computing. One definition of trust is as follows [2].

> "*Trust is both an emotional and a logical act. Emotionally, it is where you expose your vulnerabilities to people, but believing they will not take advantage of your openness. Logically, it is where you have assessed the probabilities of gain and loss, calculating expected utility based on hard performance data, and concluded that the person in question will behave in a predictable manner.*"

We have adopted the logical part of the definition, where trust means being able to predict what other people will do and what situations will occur. The working definition of trust we have adopted in this paper is that "*before I let you perform certain important actions, you need to verify that you are who you say you are, and that you are not going to do bad things to my information*".

Many different algorithms and protocols have been defined to establish trust between two interacting parties. The establishment of trust using a public key infrastructure is widely used in many applications [3]. However, this approach is not reliable in the Internet environment because it is vulnerable to attacks from malicious software [4]. Moreover, such approaches authenticate the user but not the information. Use of digital certificates bound to a specific machine is a practice industries have used to address this problem. That is, an enterprise issues a digital certificate on a *per machine* basis so that the enterprise can verify the machine, and indirectly, the person that is using the machine. This latter fact is based on an assumption that the certified machine is allocated to a particular person. Stronger, direct authentication of the user is possible through the use of PINs or two factor authentication mechanisms. Though issuing digital certificates to a particular machine has addressed the issue of authenticating the user, it still does not guarantee the authentication of data produced or the operating environment due to possible attacks from malicious software via the hostile operating environment. This issue has been recognized and addressed by bodies such as the Trusted Computing Group (TCG) [1] using a hardware-based solution. The TCG defined a Trusted Platform Module (TPM), which is a cryptographic microcontroller system that enables the TPM to be a root of trust for authenticating both the hardware and software configurations of the host computer in which the TPM is installed. The TCG remote attestation protocol is designed to establish a one-way trust relationship, where the sending host establishes a trust relationship with a receiving host. As reported in other papers [5,6,7], we have extended the TCG remote attestation protocol to allow mutual attestation of hosts in our cooperative healthcare prototype system. We further refined and proposed a *session-based mutual attestation protocol* to establish a trust relationship between interacting hosts based on an *application session* rather than for each application message exchanged [8].

The problem with all of these approaches is that the mechanism for establishing trust is tightly coupled with the hardware of a specific machine and its associated software environment. This hinders the portability and mobility of trust, as well as presenting a management problem each time a new piece of software or hardware is introduced into the host computer. For example, if an enterprise's agent needs to visit a customer's office to work on their files, then the agent must only use the issued "trusted" computer to work at the customer's office. Ideally, the agent would like to use any computer, including an untrusted customer's machine, and yet still be able to establish the strong trust relationship with the enterprise. Simple use of SSL tunnels and data encryption are not sufficient for establishing the type of strong trust relationship we envisage since the customer's machine may be compromised, for example, by malicious key-stroke logging malware or even recording the information off line for later (malicious) decryption. Establishing strong trust relationships requires that user's machines, the enterprises and the users at both ends are known to each other. Providing portability and mobility of trust and being able to use any machine anywhere in the world is not possible with existing approaches. In order to address this problem we have invented a novel technology called the *Trust Extension Device* (TED) [9]. The TED uses three basic technologies to provide mobility and portability of trust in hostile, open and untrusted environments.

- *Small portable device,* such as flash memory, is used to make the device mobile and easily portable to any platform environment.
- *Trusted Platform Module (TPM)* microcontroller is used to provide hardware-based keys for trust establishment.
- *Virtual machine technology* is used to provide an environment within an untrusted host machine to create a trusted environment.

In this paper, we outline the technologies behind TED by describing its design, a conceptual implementation using available technologies and its use in a financial application.

The rest of the paper is organized as follows. We first present a motivating scenario in the context of cooperative information systems in Section 2. In Section 3, we describe the design of the TED and its component architecture. Section 4 presents a conceptual implementation of the TED and describes its application using a financial scenario. Section 5 describes the related work of different approaches for establishing trust and then differentiates TED from them. The last section presents our analysis of the proposed technology including its shortcomings and possible future directions.

2 Context and Motivation

The motivation for this work is provided by our earlier work on eConsent where we developed a demonstrator showing how Trusted Computing technologies can be used to improve access to confidential information in a distributed healthcare environment [7] whilst protecting privacy. Trusted systems, as specified by groups such as the

Trusted Computing Group (TCG), assume that computing environments are *uniform* in terms of their operational environment, including hardware configuration, execution of a standard set of applications, operating system and facilities and procedures that allow the issue, revocation and maintenance of critical encryption keys and authorization certificates. These assumptions are applicable to a single managed enterprise infrastructure (as they need substantial care and maintenance procedures). However, in situations where the users are mobile, the computing environment is heterogeneous and the Internet provides the connectivity, management of trust between enterprises becomes overwhelmingly difficult, if not impossible. As a result, deployment and uptake of trusted secure systems based on TPM has not been as successful as first envisaged.

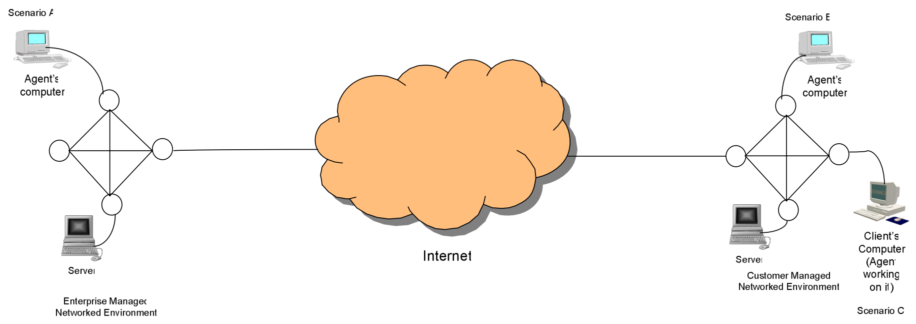

Scenario A
Agent's computer
Server
Enterprise Managed Networked Environment
Internet
Scenario B
Agent's computer
Server
Customer Managed Networked Environment
Client's Computer (Agent working on it)
Scenario C

Fig. 1. Illustration of the motivating scenario

Within the context of our work, these wider issues of trust management were first raised at the CeNTIE[1] Enterprise System Focus Group in September 2006 through a practical real-life scenario [10]. Figure 1 shows the background to the motivating scenario. An agent, working for a company, is issued a digital certificate (embedded in software or hardware) against which the agent is authenticated. The certificate is used to establish a level of trust between the agent and a company resource (e.g. a server). When the client-server link has been authenticated, customized applications and confidential client data are available for use by the agent. We discuss two possible scenarios.

The first scenario involves the agent using the machine with a preset configuration within the company's managed network (scenario A in Figure 1). A digital certificate, bound to a specific machine along with the attestation mechanism, can be used to establish trust in a controlled environment such as this. The second scenario (B) involves the agent working at the customer's managed network and uses the Internet to establish the trusted transactions with the company's server. The digital certificate, bound to a specific machine, can be used to establish the trust provided the company presets the machine with the desired configurations. That is, the agent's own machine needs to be used (scenario B in Figure 1). However, a number of issues arise with this

[1] CeNTIE is the Centre of Networking Technologies for the Information Economy, supported by the Australian Government through the Advanced Networks Program of the Department of Communications, Information Technology and the Arts.

scenario if the agent wants to use any other of the client's machines (scenario C in Figure 1):

1. The certificate is bound to a specific machine thus making it difficult for the agent to work from any other client machine. It would be impossible, for example, to use a client's machine for accessing information if the agent then tried to use the certificate issued to the agent's machine on the client's machine.
2. When an agent uses their assigned certificate, on an untrusted host machine, the security of the certificate is vulnerable to compromise by malicious software that may be running on the host machine.
3. It is possible for certificate details to be compromised in other ways, such as theft or loss. One way for a company to alleviate these potential security risks is to periodically revoke old certificates and re-issue new ones. However, this is a complex operation to manage with high overheads and especially so for large numbers of agents operating in the field.
4. Downloaded customized application's and confidential data are vulnerable to attack, since this software operates within an untrusted environment, on the host machine.

The aforementioned problem space provided the motivation for devising a mobile, portable device known as the *Trust Extension Device* (TED). The purpose of the TED is not only to provide a digital certificate embedded in the microcontroller for authentication, but also the trusted working environment that is bound to the issued digital certificate and can be instantiated and executed on heterogeneous platforms. This is critical to achieve the mobility and portability of trust on any untrusted host.

3 Trust Extension Device (TED)

TED is central to providing trust mobility, while not relying on having direct access to a secure, trusted, and managed infrastructure. A user can, for example, plug a portable device into any untrusted networked host machine to create his/her own trusted working environment, which is isolated from the host machine's environment. The created client environment appears as a virtual machine to the agent, on which they can do their work and through which they can communicate with the remote (home) server. A key part of this work is that trust mechanisms are utilised to provide attestation for all transactions between the created mobile client environment on the host machine and the remote server. The novelty in this work is that it combines trusted hardware and virtual machine mechanisms within a portable device. When an agent terminates a working session, on the untrusted machine, no remnant of client data or transactions will be traceable on the host, i.e. when the created virtual environment is terminated all data associated with it disappears.

Figure 2 presents a typical scenario of use of the proposed trust extension device. As can be seen, it involves four components: an enterprise including remote application server, trust extension device, untrusted local machine, and untrusted network connection between the local machine and remote server.

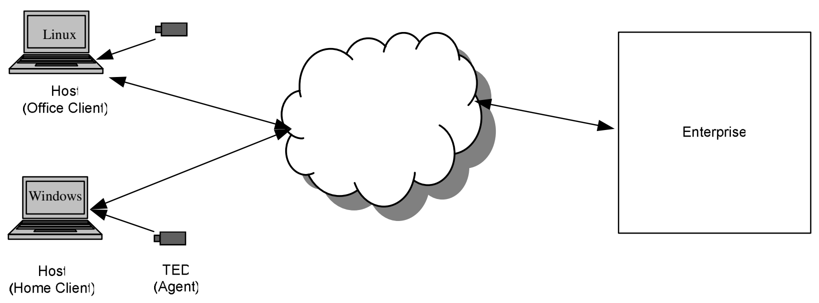


Fig. 2. A Typical TED Use-Case Scenario

We next describe a typical scenario of use for the TED. Consider an agent, working on behalf of a bank, who needs to visit a client's premises in order to clarify some financial issues. On arrival, the agent plugs the TED into the client's host machine that is running an unknown and untrusted software configuration and operating system. The TED is preconfigured so that when it is plugged into the client's machine, it automatically creates a trusted environment to deal in a predictable manner with the bank. In an ideal scenario, the trusted environment acquires control of the host machine's resources (memory, storage, I/O, and networking) for its own use in such a way that these resources, when used by the trusted environment, are isolated from the host machine's operational environment[2]. To perform tasks on behalf of a client, the agent invokes a secure (customized) application that is embedded on the TED. However, before the secure application is used, the TED is *attested* as being trustworthy. This is achieved by a mechanism that operates between an embedded trust entity on the TED and a trust verification entity running on a remote host or server. This mechanism establishes a relationship that may be understood in general terms as, *verify that you are who you say you are and that you aren't going to do bad things to my information, before we start our exchange*. Once the trust relationship is established, each subsequent application-related transactions may also be attested by this mechanism. Within this trusted context, an agent may download confidential client data, from say the bank database, operate on the data using the secure application, and then upload the modified client data back to the database. To terminate the working session, the agent may either remove the TED from the client's machine, or quit the secure application and then shutdown the TED's virtual machine, and then remove the TED from the host machine. In either case, when a session is terminated, all information exchanged between the agent and the bank is destroyed, and all the acquired resources on the untrusted host are released back to the host. It is important to note here that all information exchanged are stored in TED that expires with the particular session. We next describe the entities involve in issuing, managing, and operating TED.

[2] The shortcomings of currently available virtual machine technologies in achieving a completely isolated virtual environment are beyond the scope of the paper. This paper assumes that virtualization technology provides complete isolation of the guest environment.

3.1 TED Architecture

Figure 3 presents the TED architecture. It includes the Trusted Platform Module (TPM), virtualization software including Virtual Machine Operating System, and a secure application within a portable device (e.g., flash memory). We briefly explain these three components below and the reasons behind having them in TED.

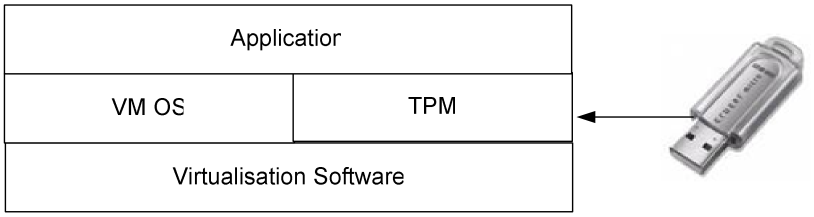


Fig. 3. TED Architecture

Trusted Platform Module (TPM)

The purpose of TPM is to provide hardware-based digital certificates for establishing trust since software-based solutions are vulnerable to malicious attacks. TPM is gaining acceptance by the computing community as a technology for establishing trust between entities (e.g. client-server) [1]. Essentially, TPM validation attests to the trustworthiness of bona fide versions of software and hardware products operating on a platform (in this case, the TED). We next review the important features of TPM in the context of TED.

The TPM is a microcontroller system with cryptographic features and data (i.e. digital keys and certificates) irrevocably "burnt" into the hardware. These features allow certain cryptographic functions to be executed only within the TPM microcontroller. Hardware and software entities outside of the TPM have restricted access to the cryptographic data on the TPM microcontroller hardware, and can only provide I/O to the TPM. Since information is burned into the TPM hardware, the TPM is deemed to be more resilient to malicious attacks than an equivalent software implementation. In brief, a TPM employs the following hardware cryptographic capabilities for verifying trustworthiness:

An *Endorsement Key* (EK) that is a public/private key-pair. The private component of the key-pair is never exposed outside the TPM. The EK is unique to the particular TPM and is generated by an enterprise (TED Issuer) "squirting" an externally generated EK into the TPM during the manufacturing process. Much of the trust value associated with the TPM comes from the fact that the EK is unique and that it is protected within the TPM at all times. This property is certified by an Endorsement Certificate (or Credential).

An *Endorsement Certificate* (or credential) that contains the public key of the EK. The purpose of the Endorsement Certificate is to provide attestation that the particular TPM is genuine, i.e. that the EK is protected.

An *Attestation Identity Key* (AIK) that is used to provide platform authentication to a service provider (or verifier). AIKs are created using certificates available within the TPM. AIKs are always bound to the platform and can be used to provide attestation to the platform's identification and configuration. The verifier attests to the TPM's authenticity by proving facts about the TPM. Goals of the proof are that the TPM is verified as the owner of the AIK and the AIK is tied to both a valid EK Certificate and a valid Platform Certificate.

A *Platform Certificate* (or Credential) that is provided by the platform vendor and provides attestation that the security components of the platform are genuine.

In Summary, the purpose of the TPM in the TED is to establish a trust relationship between client-server entities where the goals of trust are to ascertain that:

- The TPM is the genuine owner of certain cryptographic keys and certificates, i.e. the TPM has not been tampered with.
- The software operating within the (created) trusted environment is genuine; which includes validating secure applications and the operating environment.
- If a TED is lost or stolen, then the issuer of the TPM revokes the manufacturers TPM credentials.

Virtualization Software and VM OS

The purpose of virtualization software in TED is two-fold. First, the virtualization software is the key to mobility and portability as it provides an operational environment in any untrusted host machines running under different platforms. Second, it is expected to provide a trusted environment by isolating the operational environment of the TED from the host machine environment. We next review some important features of Virtual Machines in the context of TED.

There is a recent trend of increasing use of Virtual Machines (VMs) operating on a shared, common hardware platform [11,12]. A VM provides familiar functions and services expected by computer programs (i.e. CPU, memory, I/O, networking), but does so using software rather than hardware. When the state of a process or system is virtualized, it is separated from a specific piece of physical hardware. This means that a virtual process or virtual system can be temporarily associated with specific physical resources and that this association can change over time. Motivations for using VM technology in TED are that it:

- Decouples computer software design (e.g. application code) from the evolution and diversity of the underlying computer hardware and operating systems. Consequently, the same application code can be used on any system that supports the appropriate VM.
- Allows applications to be ported on different computer platforms therefore enabling programs to be easily taken by the user to different physical locations.

There are two types (Type I and Type II) of virtualization. A simple representation of a Type I VM is shown in

Figure 4 (a), where virtualising software (the VM Monitor - VMM) is placed between the underlying machine and conventional software. In this example, the virtualising software translates the hardware Instruction Set Architecture (ISA) so that the conventional software "sees" a different ISA from the one supported by the hardware. The virtualisation process involves, (1) mapping virtual resources, e.g., registers and memory, to real hardware resources, and, (2) using real machine

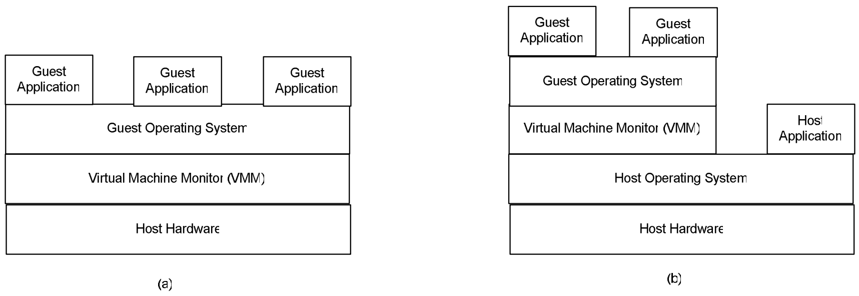


Fig. 4. Virtual Machine Architectures: (a) Type I (b) Type II

instructions to emulate the virtual machine ISA. The underlying platform is known as the "host" and the software that runs in the VM environment is the "guest".

For TED, we utilize a Type II VM for the purpose of creating a secure trusted environment as shown in shown in Figure 4 (b). This enables mobility of TED to heterogeneous platforms. A brief outline of a full VM follows. There are some limitations on TED while using a Type II VM, which we will discuss in a later section.

In some important situations, the guest and host systems do not have a common Instruction Set Architecture (ISA) and operating system. For example, the Apple PowerPC-based systems and Windows PCs use different ISAs and different operating systems. Because software systems are so closely tied to hardware systems, the purchase of multiple platforms is required to support commonly used applications. This situation motivates system VMs, where a complete software system, including Operating System (OS) and applications, is supported on a host system that runs a different OS and ISA. These are called Type II VMs because they virtualize the entire host platform. Since the guest and host ISAs are different, both the guest OS and guest application code require emulation; i.e. the virtualizing software must, 1) emulate the entire hardware environment, 2), control the emulation of all the instructions, and 3) convert the guest ISA operations to equivalent OS calls made to the host OS.

For Type II VMs, the most common implementation method is to place the virtualizing software and guest software on top of a conventional host OS that runs on the host hardware, as shown in Figure 4 (b), i.e. a guest Linux system operating over a host Windows system. It is as if the virtualizing software, the guest OS and guest applications are one large application implemented on the host OS and host hardware. At the same time, the host OS continues to run applications compiled for the native ISA. The shortcoming of such a virtual machine, as we discussed earlier in a footnote, is that it does not provide complete isolation of the guest environment as envisaged in TED. We are investigating alternative approaches to overcome this shortcoming.

Secure Application

TED can be used in a range of applications, from financial transactions, to distributed collaborations. The secure application is a thin client layer of the TED architecture which is loosely coupled with underlying components. The secure application can be developed using existing security technologies such as SSL. The TED is designed in such a way that a developer can design and write an application that can be loaded into the TED. The secure application is specially customised for use by a mobile user (e.g. company agent). The trusted application utilises the TPM on the TED to establish trust. Trust is established between a trusted application and a specific server deployed by the issuer of the TED (the company). The detailed description of how a secure application works is given in Section 3.3.

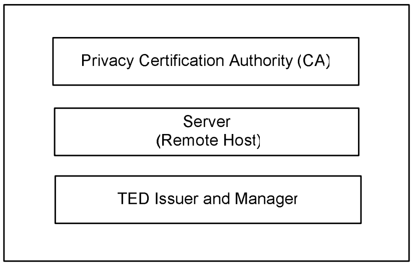


Fig. 5. Basic Components in an Enterprise Architecture

3.2 Enterprise System Architecture

An enterprise needs three basic components for TED to be operational as shown in Figure 5.

TED Issuer and Manager that is responsible for generating digital keys, manufacturing, issuing and revoking the TED.

Privacy Certifying Authority that is responsible for verifying the TED.

Application server that is responsible for deploying any related enterprise applications.

It is important to note that these three tasks can be performed by three different entities. For example, a bank can provide its own enterprise applications, but use a trusted third party manufacturer and certifying authority to perform the other two tasks. However, for our discussions in this paper, without loss of generality, an enterprise is assumed to perform all these tasks. We next describe these three components in detail.

TED Issuer and Manager

It is envisaged that the manufacture of the TED will be authorized by an enterprise (such as a banking institution). The role of the enterprise in the manufacture of the TED is to supply the necessary credentials as shown in Figure 6 that include cryptographic keys for each TED, and in particular, the Endorsement Key pair for the Endorsement Credential where the Endorsement Credential is embedded into the TPM component of the TED. In our current architecture, we assume that the TED issuer and manager within the enterprise will assume this role. That is, a single enterprise can authorise many TEDs through the TED issuer and manager. The TED manager will sign the credentials using its cryptographic private key. The TED manager, therefore, will generate, for a single TED, the following data:

TED Credential containing data that identifies the person/client to whom the TED is issued by the enterprise. The details of the client are signed by the enterprise; in our enterprise architecture, this is done by the TED manager.

Endorsement Credential includes the public part of the endorsement key that is unique to each TED. The TPM manufacturer signs the endorsement key. This is done by the TED manager in our enterprise architecture.

Platform Credential includes the TED's operating environment consisting of VM software and VM OS. In our architecture, the TED manager signs the details of the platform. It is possible to have an independent third party supplying the platform description.

Validation Credential includes service component descriptions consisting of their digests that are loaded into the TED. One could have an independent validation manager. In our simple enterprise architecture this is also achieved by the TED manager.

The TED Manager or issuer manufactures the TED with the credential details explained above so that it can be used to establish the trust relationship as described in a later section.

Privacy CA

The TCG uses a trusted third party, the privacy certification authority (Privacy CA), to verify and authenticate the TPM. The same concept is used in TED which works as follows. Each TED is issued with the credentials including an RSA key pair called the Endorsement Key (EK). The Privacy CA is assumed to know the credential details along with the public parts of the Endorsement Keys of all TEDs. That is, the TED manager supplies the credential details to the Privacy CA. Whenever a TED needs to communicate with the enterprise, it generates a second RSA key pair, called an Attestation Identity Key (AIK), sends an identity key certification request to the Privacy CA, which contains, (a) an identity public key, (b) a proof of possession of identity for the private key, and (c) the endorsement certificate containing the TED's endorsement public key as shown in Figure 7. The privacy CA checks whether a TED issuer has signed the endorsement certificate. If the check is successful, the privacy CA returns an identity certificate encrypted with the TED's endorsement public key. The TED can then provide this certificate to the application server to verify and authenticate itself with respect to the AIK. If the TED is reported as stolen or lost, the Privacy CA can compute the corresponding public key and tag it as a rogue TED.

Application Server

The exact nature of the server application and its software architecture depends on the specific application. We describe one such application server later on in our prototype implementation section.

3.3 Trust Establishment Protocol

Trust is established by use of the TED's Endorsement Credential of the TPM. The Endorsement Credential contains the public part of a customized cryptographic key-pair generated by the enterprise. The Endorsement Credential is obtained from the trusted application of the TED's TPM as instances of Endorsement Certificates. In

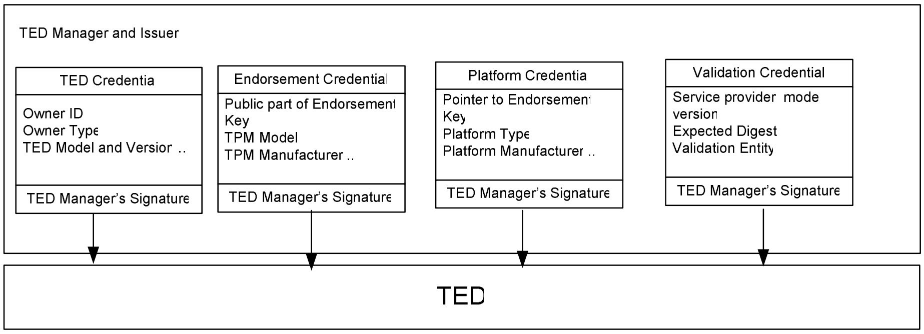


Fig. 6. TED's Credentials managed by TED issuer and manager

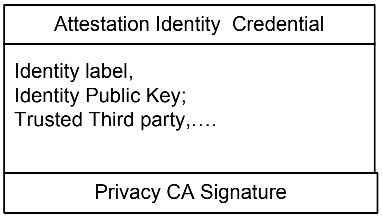


Fig. 7. Attestation Identity Credential

addition, the Endorsement Credential is digitally signed by the private part of the enterprise's (TED manager's) own cryptographic key-pair; this implies that the Endorsement Certificate is also signed by the private part of the TED manager's cryptographic key-pair.

To establish trust, the trusted application initiates the generation of an Attestation Identity Key (AIK); the AIK is generated inside the TED's TPM. The trusted application then sends the public part of AIK, the Endorsement Credential to the TED's TPM, and the TED's credential to the trust verifier (i.e. Privacy CA). From this data, the trust verifier can determine (a) whether the Endorsement Credential is as expected and (b) the originating TED's TPM that generated and sent the Endorsement Certificate.

The trust verifier then generates an AIK Certificate, which is then sent back to the trusted application. The AIK Certificate vouches for the genuineness of the TED's TPM, i.e. the AIK Certificate ensures that the AIK came from a particular TPM. The AIK and AIK Certificate are used by the trusted application for all future enterprise transactions. The AIK is necessary because the public part of the Endorsement Key cannot be used to perform cryptographic operations on data outside of the TPM, but the AIK can.

The trusted application can use the AIK and AIK Certificate in cryptographic operations from this point onwards. Some examples include establishing a secure connection with a server (via TLS, SSL or even possibly IPSec), or just to encrypt data to send to the server. The trusted application can also send an integrity measurement (cryptographic hash) of itself before performing any operations with the

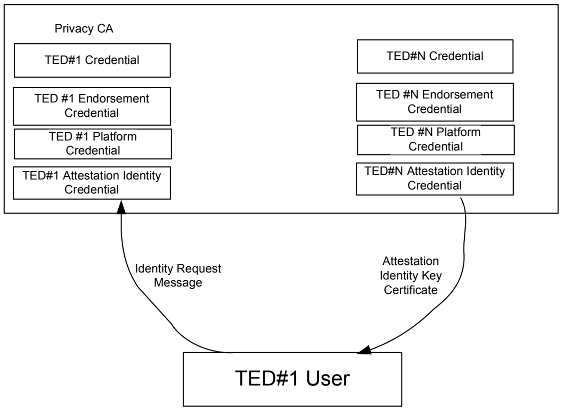

Privacy CA
TED#1 Credential
TED #1 Endorsement Credential
TED #1 Platform Credential
TED#1 Attestation Identity Credential
TED#N Credential
TED #N Endorsement Credential
TED #N Platform Credential
TED#N Attestation Identity Credential
Identity Request Message
Attestation Identity Key Certificate
TED#1 User

Fig. 8. Process of Issuing AIK Certificate to establish the trusted device

server. The TED manager then uses digests in validation credentials to validate the service components.

The above process of generating the AIK, sending the AIK with credentials to the server, obtaining an AIK Certificate and sending an integrity measurement itself is known as *remote attestation*. The first three steps are related for establishing the trusted device and the last step is for the trusted environment. This can be performed either on a per-transaction basis [7] or a per-connection basis [8]. Per-transaction refers to the process where for every operation the trusted application performs remote attestation, or for every instance in which the trusted application needs to communicate with the server, the trusted application performs remote attestation. Per-connection attestation is where remote attestation is performed only once during the start-up of the connection to the server. Figure 8 illustrates the concept of establishing a trusted device for a single TED.

4 Prototype Implementation

We have developed a prototype system to demonstrate the concept of TED. Our prototype implementation follows the enterprise architecture described earlier. Figure 9 shows the implementation architecture for TED. Following is a description of the

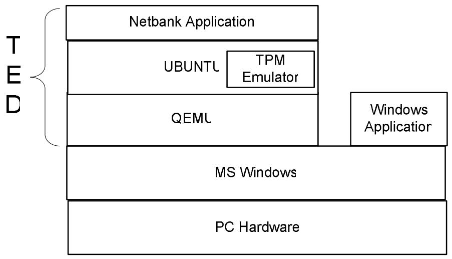


Fig. 9. Implementation Architecture

different components of the architecture, and how they work together in order to enable mobility and portability of trust for a financial application.

TPM Emulation

There are a number of commercially available options for realizing the TPM in the TED, any one of which is applicable for use. The actual TPM hardware can be either TPM version 1.1 or TPM version 1.2. It is anticipated that the features required by the TED from the TPM are accessible in both TPM version 1.1 and TPM version 1.2 hardware. Features of a TPM (after manufacture) required by the TED are: Obtaining ownership of the TPM; Access to obtain the Endorsement Certificate; Generation of an Attestation Identity Key (AIK); Generation of an Identity Request message; Loading of an Attestation Certificate; and Generating and storing cryptographic hashes.

Our prototype system utilized a TPM software emulator [13]. In order for the trusted application to access the features listed above, the host operating system requires a Device Driver of the TED's TPM and APIs to access TPM features. The device driver can either be supplied by the manufacturer of the TPM or it can be a generic device driver (included in the guest operating system or from a third-party supplier). Likewise, the APIs needed to access the TPM hardware can either be supplied by the manufacturer of the TPM or from sources such as the guest operating system or from a third party.

In our prototype, the TPM device driver and APIs are from a third-party supplier. Specifically, the APIs are from IBM's TrouSers TSS project (version 0.2.7) [14] and jTssWrapper (version 0.2.1) [15].

QEMU

On plugging the TED into a host machine, the host machine's operating system will sense the TED as a USB storage device which invokes the host machine to execute the TED's virtual machine. The TED's virtual machine then instantiates an isolated trusted environment on the host machine; this trusted environment is the virtual machine that a user interfaces to. TED's virtual machine is customized to obtain certain types and amounts of host machine resources, such as specific amounts of memory, general I/O (i.e. keyboard, mouse and video), and network access.

The benefit of having a virtual machine execute on connection is that it does not require any installation of special software by the host machine; all the software required for the virtual machine to execute is contained in the storage area of the TED. In addition, the TED virtual machine directs all TPM related communications and data from the trusted application and trusted environment to the TED's TPM and not the host machine's TPM, should it have one. Furthermore, the current implementation of the TED's virtual machine does not require re-booting of the host machine; i.e. TED's virtual machine and the host machine's operating system can co-exist in parallel but are isolated from each other.

In our prototype system, QEMU, version 8.2.0 for Microsoft's Windows XP, [16] is used as TED's virtual machine. QEMU is an open source processor emulator. The primary intention of the QEMU is to run one operating system on top of another, such as Linux on Windows. QEMU has two operating modes, full system emulation and user mode emulation. For our implementation, we only use the full system mode. Full system emulation (of a PC) includes emulation of the processor and various peripherals in order to launch a guest operating system. As a full system emulator QEMU can run an unmodified guest operating system, such as GNU/ Linux or Windows, and all its applications in a virtual machine. QEMU can also run on several host operating systems such as Windows, Linux and Mac OS X where the host and guest CPUs can be different. It is important to note here that such a virtual machine does *not* provide complete isolation of the guest environment from the host machine. This is a technological shortcoming in achieving the "trusted" guest environment as envisaged in the concept of TED. This observation has led us to further work on obtaining complete isolation of the guest environment from the host environment without having any effect on the portability and mobility of trust.

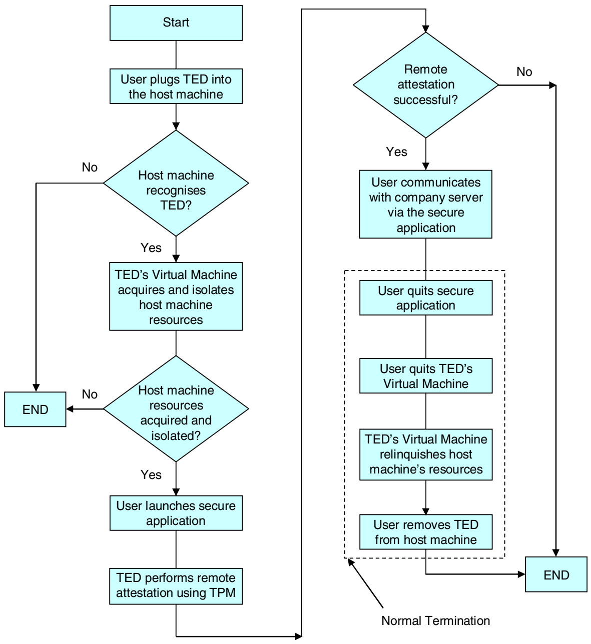

Start
User plugs TED into the host machine
Host machine recognises TED?
No
Yes
TED's Virtual Machine acquires and isolates host machine resources
Host machine resources acquired and isolated?
No
END
Yes
User launches secure application
TED performs remote attestation using TPM
Remote attestation successful?
No
Yes
User communicates with company server via the secure application
User quits secure application
User quits TED's Virtual Machine
TED's Virtual Machine relinquishes host machine's resources
User removes TED from host machine
END
Normal Termination

Fig. 10. The TED Runtime Flow Diagram

UBUNTU

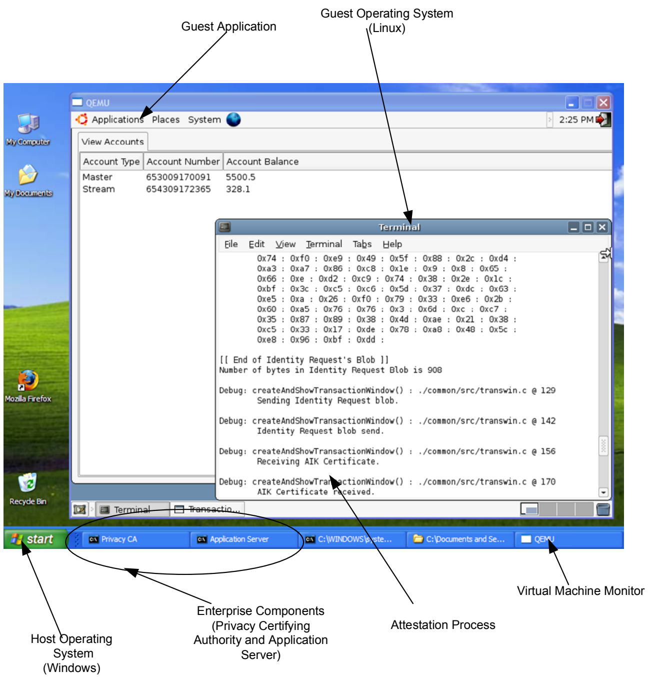
In our prototype system, QEMU creates an isolated trusted environment in the host machine's Microsoft's Windows XP operating system. The guest operating system is a customised Ubuntu 6.06 i386 GNU/Linux distribution.

The TED's trusted environment, customised Ubuntu 6.06 i386 GNU/Linux and application(s), are stored as a disk image file on the TED's storage area. This customised operating system is presented to the user, as the trusted environment, after the QEMU has successfully acquired and isolated the resources from the host machine. QEMU does not require the host machine to be re-booted; i.e. QEMU allows for co-existence of both the host machine's operating system and its applications, in parallel with the trusted environment.

NetBank Application

In our prototype system, we have developed a client for a web-based banking application. The client application performs simple data retrieval from a server (described earlier) across a TCP/IP network. Before data retrieval can begin, the client application first performs remote attestation with a Privacy-CA server across the network. The client application is only allowed to perform data retrieval when the remote attestation operation is successful.

The client application was developed for the operating system of the TED; in this case, the Ubuntu GNU/Linux distribution (version 6.06 for IA-32 architecture). The client application used GTK+ for its GUI library and traditional UNIX network sockets for its networking capabilities.



Account Type	Account Number	Account Balance
Master	653009170091	5500.5
Stream	654309172365	328.1

```
        0x74 : 0xf0 : 0xe9 : 0x49 : 0x5f : 0x88 : 0x2c : 0xd4 :
        0xa3 : 0xa7 : 0x86 : 0xc8 : 0x1e : 0x9 : 0x8 : 0x65 :
        0x66 : 0xe : 0xd2 : 0xc9 : 0x74 : 0x38 : 0x2e : 0x1c :
        0xbf : 0x3c : 0xc5 : 0xc6 : 0x5d : 0x37 : 0xdc : 0x63 :
        0xe5 : 0xa : 0x26 : 0xf0 : 0x79 : 0x33 : 0xe6 : 0x2b :
        0x60 : 0xa5 : 0x76 : 0x76 : 0x3 : 0x6d : 0xc : 0xc7 :
        0x35 : 0x87 : 0x89 : 0x38 : 0x4d : 0xae : 0x21 : 0x38 :
        0xc5 : 0x33 : 0x17 : 0xde : 0x78 : 0xa8 : 0x48 : 0x5c :
        0xe8 : 0x96 : 0xbf : 0xdd :

[[ End of Identity Request's Blob ]]
Number of bytes in Identity Request Blob is 908

Debug: createAndShowTransactionWindow() : ./common/src/transwin.c @ 129
        Sending Identity Request blob.

Debug: createAndShowTransactionWindow() : ./common/src/transwin.c @ 142
        Identity Request blob send.

Debug: createAndShowTransactionWindow() : ./common/src/transwin.c @ 156
        Receiving AIK Certificate.

Debug: createAndShowTransactionWindow() : ./common/src/transwin.c @ 170
        AIK Certificate received.
```

Fig. 11. A screenshot of the running prototype

Application Server

We have implemented a multi-threaded bank server as a java application using Eclipse 3.1 IDE and jdk 1.5 runtime environments. The bank server communicates with bank clients via sockets. The multi-threaded bank server listens to the socket for a bank client to make a connection request. The bank client knows the hostname where the server is running and the port number on which the server is listening. After a connection is established the bank server creates a thread that can receive client requests and subsequently process them. The current bank server provides two services. It can validate the bank client by checking username/password. It can also return account balances of the authenticated bank clients. The bank server also maintains a database for the list of valid bank clients and their account balances.

How does TED work?
An enterprise can issue one or many TEDs. Since the enterprise is involved during the manufacturer of the TEDs, the enterprise has all the credential information of the TEDs it issues. In addition, the enterprise has two additional components, the Enterprise's Application Server and the Enterprise's Privacy Certificate Authority.

The Enterprise's Server is used when a TED's application requires a service from the enterprise. The Enterprise's Privacy Certificate Authority is used to perform remote attestation whenever a TED connects to the enterprise's network.

The flow diagram shown in Figure 10 captures the sequence of events (from the point of view of a company agent) from the time that the TED is plugged into an untrusted host to termination of a working session and removal of the TED. Figure 11 shows a screenshot capture while running the TED's NetBank application in a Windows XP host.

5 Related Work

One of the methods commonly used to establish trust is the use of public-key infrastructure. In this approach, when a local host is challenged by a remote host, it can verify the identity and trustworthiness of the local host by verifying the signature on local host's public key [23]. This method however does not take into account the integrity of the operating environment. This shortcoming can be addressed by using a software stack to measure and verify the integrity of co-operating systems [19, 20, 21]. However, one of the most critical disadvantages of using software-based solutions is that the hardware which hosts the software stack can be stolen or the hardware can be hacked to monitor the behavior of the stack.

In recent times, hardware-based techniques have gained popularity due to the fact that hardware is relatively harder to hack than software. The hardware most relevant to our work embeds TCG remote attestation [1] that allows distributed remote hosts to verify each other by sending a snapshot of the current state of the platform configuration. Several problems have been identified related to remote attestation due to (a) the difficulty of measuring a platform's configuration accurately in today's complex systems, and (b) the difficulty of architecting a trusted central authority within this untrusted world, where totally independent systems communicate with each other in an autonomous manner [22, 23]. These problems are driving further research activities in remote attestation.

One of the approaches that use virtual machines is Terra [26]. The key premise that Terra builds on is a trusted virtual machine monitor (TVMM) that allows Terra to partition the platform into multiple isolated VMs. The TVMM provides a narrow interface for attestation. It first generates a certificate that forms the basis of attestation that contains the hash of a VM, TVMM's public key and any other application data used for authenticating the VM. The remote party retrieves this certificate to check the validity of a VM. However, these techniques lack portability and mobility of trust as the mechanism of establishing trust is attached to a specific machine.

One of the commercially available mechanisms developed to address the mobility and portability of trust is the eToken [17]. An eToken is a true reader-less smartcard that can be taken from one workstation to another. Though eToken provides mobility and portability of trust that includes secure storage and a robust file system, the mechanism still uses a software-based mechanism but does not provide a trusted environment like that of the TPM-based solution. On the other hand, SoulPad [18] provides the portability of an operational environment without an underlying trust mechanism. In many aspects, TED provides an integrated hardware-based solution for portability and mobility of a trusted environment which encompasses the mobility and portability of the digital certificate aspects of the eToken, the hardware-based trust establishment mechanism of TPM, the portability and mobility of the operational environment of SoulPad and the isolation of virtual machine of Terra. Therefore TED enables the portability and mobility of isolated hardware-based trusted environments.

6 Conclusions

With TED we have provided a mechanism of providing mobility and portability of trust so that users can create and use a trusted environment on any untrusted host machine. Our approach uses the TPM for providing hardware-based digital certificates and virtual machine technologies for providing an isolated environment. We have implemented the concept on a financial application using the currently available software technologies. The implementation also provides us insights into the isolation requirements of the virtual machine in order to achieve the desired environments for TED. We found that the level of isolation required for the virtual machine, in the host machine, can be obtained only if the host machine or TED is built for such proposes. However, the complete isolation of the guest environment from the host environment on any host machine is not possible using the currently available virtualization technologies. This observation has led into further work on obtaining complete isolation of the guest environment from the host environment without compromising on portability and mobility. Our initial work towards this focuses on the development of secure I/O within TED using a bootable TED, which provides the security at the cost of “flexibility” and “portability”. We also demonstrated the realization of TED using TPM enabled laptops. Our future work includes developing a security solution for TED without compromising the portability and mobility of trust.

References

1. TCG specification v1.1, https://www.trustedcomputinggroup.org/specs/TPM/
2. http://changingminds.org/explanations/trust/what_is_trust.htm
3. Satizábal, C., Páez, R., Forné, J.: Relationships: from a Hybrid Architecture to a Hierarchical Model. In: Proceedings of the First International Conference on Availability, Reliability and Security (ARES 2006)
4. Yang, R., He, L., Yang, S., Gary, F., Liu, F., Chang, J., Guo, M.: The Value of Hardware-Based Security Solutions and its Architecture for Security Demanding Wireless Services. Security and Management , pp.509–514 (2006)

5. Nepal, S., Zic, J., Kraehenbuehl, G., Jaccard, F.: A trusted system for sharing patient electronic records in autonomous distributed healthcare systems. International Journal of Healthcare Information Systems and Informatics 2(1), 14–34 (2007)
6. Nepal, S., Zic, J., Jaccard, F., Krachenbuehl, G.: A Tag-based Data model for privacy-preserving medical applications. In: Proceedings of EDBT IIHA Workshop, Munich, Germany, pp. 77–88 (2006)
7. Nepal, S., Zic, J., Krachenbuehl, G., Jaccard, F.: Secure Sharing of Electronic Patient Records, 1st European Conference on eHealth, pp. 47–58. Fribourg, Switzerland (2006)
8. Jang, J., Nepal, S., Zic, J.: Establishing a Trust Relationship in Cooperative Information Systems. In: Meersman, R., Tari, Z. (eds.) Proceedings of Cooperative Information Systems (CoopIS) 2006 International Conference. LNCS, vol. 4275, pp. 426–443. Springer, Heidelberg (2006)
9. Nepal, S., Zic, J.: A Portable Trusted Device, Provisional Australian Patent, September (2006)
10. http://www.ict.csiro.au/page.php?did=14
11. http://www.xensource.com/
12. http://www.vmware.com/
13. http://developer.berlios.de/projects/tpm-emulator/
14. http://trousers.sourceforge.net/
15. http://trustedjava.sourceforge.net/jtss/javadoc/
16. http://fabrice.bellard.free.fr/qemu/about.html
17. http://www.aladdin.com/eToken/
18. Caceres, R., Carter, C., Narayanaswami, C., Raghunath, M.T.: Reincarnating PCs with Portable SoulPads. In: Proc of ACM/USENIX MobiSys, pp. 65–78 (2005)
19. Kennell, R., Jamieson, L.H.: Establishing the genuinity of remote computer systems. In: Proceedings of the 11th USENIX Security Symposium, USENIX, August (2003)
20. Seshadri, A., Perrig, A., van Doorn, L., Khosla, P.: SWAtt: SoftWare-based Attestation for embedded devices. In: Proceedings of IEEE Symposium on Security and Privacy, (May 2004)
21. Monrose, F., Wyckoff, P., Rubin, A.D.: Distributed execution with remote audit. In: ISOC Network and Distributed System Security Symposium, pp. 103–113 (1999)
22. Haldar, V., Franz, M.: Symmetric Behavior-Based Trust: A New Paradigm for Internet Computing. In: New Security Paradigms Workshop (September 2004)
23. Reid, J., Juan, M., Nieto, G., Dawson, E., Okamoto, E.: Privacy and Trusted Computing. In: Mařík, V., Štěpánková, O., Retschitzegger, W. (eds.) DEXA 2003. LNCS, vol. 2736, p. 383. Springer, Heidelberg (2003)
24. AMD platform for trustworthy computing. WinHEC 2003, http://www.microsoft.com/whdc/winhec/papers03.mspx, Sept. 2003
25. Millen, J.K., Wright, R.N.: Reasoning about Trust and Insurance in a Public Key Infrastructure, 13th IEEE Computer Security Foundations Workshop(CSFW), 2000, pp. 16–22 (2000)
26. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: A virtual machine-based platform for trusted computing. In: Proceedings of Symposium on Operating System Principles (SOSP) (October 2003)

Organizing Meaning Evolution Supporting Systems Using Semantic Decision Tables

Yan Tang and Robert Meersman

Semantic Technology and Application Research Laboratory (STARLab),
Department of Computer Science,
Vrije Universiteit Brussel, Pleinlaan 2 B-1050 Brussels, Belgium
{yan.tang, robert.meersman}@vub.ac.be

Abstract. DOGMA-MESS (Meaning Evolution Support System) is a system and methodology for supporting scalable, community-grounded ontology engineering. It uses a socio-technical process of meaning negotiation to tackle the scalability problems in ontology engineering. In order to improve the effectiveness of DOGMA-MESS, we adopt the idea of Semantic Decision Table (SDT). An SDT contains semantically rich decision rules that guide DOGMA-MESS micro-processes. It separates the decision rules from DOGMA-MESS. SDT with different decision rules results in different final decisions, which can be evaluated. In this paper, we illustrate how SDTs are used and apply our approach in the domain of Human Resource Management.

Keywords: ontology, Meaning Evolution Support System, Semantic Decision Table, DOGMA-MESS.

1 Introduction

Nowadays, a vast amount of ontology capturing methodologies and tools are available. However, scalable ontology engineering is hard to do in an interorganizational setting, where there are many pre-existing organizational ontologies and rapidly evolving collaborative requirements. A viable methodology requires not building a single, monolithic domain ontology, but supporting *many* domain experts in increasingly building ontologies. Accordingly, *DOGMA-MESS* (Meaning Evolution Support System) methodology [4] is developed for scalable community-grounded ontology engineering.

A core activity of DOGMA-MESS is to reach final consensus through a careful and gradual process of *meaning negotiation* [3]. Reaching consensus on ontologies by means of meaning negotiation tackles the scalability problem. As a key mechanism of meaning evolution systems, meaning negotiation is integrated in the organizational ontology alignment of DOGMA-MESS. In order to capture the behaviors of domain expert community in a natural collaborative decision making manner, we adopt *Semantic Decision Tables* (SDT, [16]), with which we try to improve the effectiveness of MESS processes within a large community.

SDT is not restricted to a specific system, such as DOGMA-MESS mentioned in this paper. Instead, we use DOGMA-MESS as an example to demonstrate how SDT can be used to support group decisions. There are two main advantages of applying SDT in such systems. One is the *flexibility* of *organizing group decision rules*, for SDT separates the formally agreed decision rules from the systems. The other is the *ability* of *analyzing* different *decision rules*. The tabular reports generated by SDT are easy for the non-technical users[1] to choose the decision rules that meet their needs.

The paper is organized as follows: we introduce the notion of Semantic Decision Table (SDT) in section 2. We give an overview of DOGMA-MESS using SDT (section 3). The experiments on using different decision rules for SDT are demonstrated in section 4. In section 5, we discuss and list related work. We conclude and open future work in section 6.

2 Semantic Decision Tables

Semantic Decision Tables (SDT, [16, 17]), which contains semantically rich decision rules, constitutes the kernel decision knowledge to reach consensuses in a large community. More precisely speaking, it *shares* the most important decision knowledge within a group decision session (e.g. a meaning negotiation process).

2.1 Decision Table and Semantic Decision Table

An SDT uses the tabular presentation as a decision table does. Following the *de facto* international standard [2], three groups of information in a decision table are the basic decision table elements: the *conditions*, the *actions* (or *decisions*), and the *rules* that describe which actions might be taken based on the combination of conditions. A *condition stub* contains a statement of a condition and each *condition entry* indicates the relationship between the various conditions in the condition stub. Each *action stub* has a statement of each action to be taken and the *action entries* specify whether (or in what order) the action is to be performed for the combination of conditions that are actually met in the rule column.

Table 1. A simple decision table example

Condition	Column 1	…	Column n
Driver's license	With	…	Without
…	…	…	…
Action / Decision			
a driver	*	…	
not a driver		...	*

Table 1 demonstrates a very simple decision table to check whether a person can be a driver or not. The table element "Driver's license" is a condition stub. The condition entry "with", together with "Driver's license", forms a condition. The table

[1] In DOGMA-MESS, the users are the domain experts who contribute to the ontology construction.

element “a driver” is an action stub. The action entry “*” and the action stub “a driver” constitute an action (or decision). Column 1 is an example of a decision.

What makes SDT different from a traditional decision table is the *semantics*. Unlike traditional decision tables, the concepts, variables and decision rules are explicitly defined in an SDT. There are two kernel constituents in an SDT: 1) the *behavior* of the community in the (external) *collaborative* environment(s); and, 2) the semantics model designed by the group when *observing* the real-word of decision making problems. To study the behavior of the community is beyond the paper focus. Readers who are interested in it may read our recent papers [18, 19]. We focus on how to model the semantics of an SDT in the next subsection.

2.2 Modeling Semantic Decision Tables

An ontology is a *semiotic representation* of *agreed conceptualization* in a subject domain [7, 8]. We use ontologies to store the SDT semantics. Note that the ontologies are used for two purposes in this paper. One is the result of the DOGMA-MESS activities. The other is to store the group decision rules to guide the DOGMA-MESS process. The ontology used to store SDT semantics is to serve the latter purpose.

SDT is modeled based on the DOGMA (Developing Ontology-Grounded Methods and Applications) framework [15], which was designed as a methodological framework inspired by the tried-and-tested principles of modeling conceptual databases. In the DOGMA framework one constructs (or converts) ontologies by the *double articulation* principle: the ontology base layer that contains a vocabulary of simple facts called *lexons*, and the *commitment* layer that formally defines rules and constraints by which an application (or “agent”) may make use of these lexons.

A lexon is a quintuple $< \gamma, t_1, r_1, r_2, t_2>$, where γ is a context identifier which is assumed to point to a resource, and which serves to disambiguate the terms t_1, t_2 into the intended concepts. r_1, r_2, which are “meaningful” in γ, are the roles referring to the relationships that the concepts share with respect to one another. For example, a lexon <*γ, Driver’s license, is issued to, has, Driver*>[2] explains a fact that “a driver’s license is issued to a driver”, and “a driver has a driver’s license”.

A *commitment*, which corresponds to an explicit instance of an intentional logical theory interpretation of applications, contains a set of rules in a given syntax and describes a particular application view of reality such as the use by the application of the (meta-) lexons in the ontology base. This describing process is also called ‘*to commit ontologically*’. The commitments need to be expressed in a *commitment language* that can be interpreted, such as in [5].

Suppose we have a lexon <Driver’s license, is issued to, has, Driver>, which has the constraints as “*one* driver’s license is accepted by *at most one* driver”. We apply the uniqueness constraints *UNIQ* on the lexon written as below:

```
p1 = [Driver's license, is issued to, has, Driver]:
UNIQ (p1).[3]
```

[2] In this paper, we do not focus on the discussion of the context identifier γ, which is omitted in other lexons. E.g. <*γ, Driver’s license, is issued to, has, Driver*> is thus written as <*Driver’s license, is issued to, has, Driver*>.

[3] The syntax of the formalized commitment and the examples can be found at: http://www.starlab.vub.ac.be/website/SDT.commitment.example.

Although an SDT contains SDT lexons and SDT commitments, SDT itself is *not* an ontology. It is because each SDT is used for a *specific* application or task. The SDT commitments can contain both static, ontological axioms and temporal, changeable rules. Different algorithms used in DOGMA-MESS need to be *committed* by a community, the result of which is a set of SDT commitments (section 4).

3 DOGMA-MESS

DOGMA-MESS (Meaning Evolution Support System) is a state-of-art system built for *scalable ontology engineering* [4]. It helps the communities of practice consisting of the stakeholders from different organizations to define the ontologies that are relevant to their joint collaborative objectives. As the ontology grows when the collaborative communities evolve and learn; it is necessary to have a systematic method of ontology evolution and alignment.

DOGMA-MESS decomposes the macro-processes, such as ontology *alignment*, ontology *merging* and ontology *versioning*, into a manageable combination of several micro-processes. It uses the socio-techniques of meaning evolution support systems to analyze formal concept definitions. The relevance of eliciting and applying the ontological knowledge of evolution is at the heart of DOGMA-MESS.

During the evolution of an ontology, the number of the ontological definitions (i.e. the concepts in a domain ontology) grows while the collaborative communities evolve and learn. The process of modifying ontologies is very expensive because the domain experts need to reach a final agreement for each new (or modified) concept. In order to solve this problem, *meaning negotiation*, which is defined as "a mechanism for reaching the *appropriate* amount of consensus on *relevant* conceptual definitions" in [3], is designed as the key focus. With the help of the meaning negotiation techniques, stakeholders in the community are able to define problems, analyze requirements and analyze/refine communication patterns for the community-driven ontology evolution.

Dependant templates designed by the stakeholder communities are applied to check the relevance between new concepts and concepts in an existing ontology during each meaning negotiation process. In [13], authors use conceptual graphs [14] to design these templates.

In the early papers, de Moor et al. [4, 5] address the questions of how to model the process of DOGMA-MESS and how to integrate the techniques of *meaning negotiation* into DOGMA-MESS processes. The issue of capturing the community's behavior during DOGMA-MESS processes was not yet covered. In this paper, we focus on using SDT [16] to support building group consensuses for DOGMA-MESS. More precisely speaking, we study the community's behaviors during the DOGMA-MESS process of *defining new relevant concepts* and specify them within an SDT. By using such an SDT in a natural collaborative decision making manner, we try to improve the effectiveness of DOGMA-MESS.

DOGMA-MESS, which is a collection of meaning evolution support systems, distinguishes ontologies at several levels. This kind of hierarchical ontology structure designates the dependencies between the concepts in layered ontologies. Recently, modeling layered ontologies has been studied so far. The scalable ontology model we will describe focuses on neither the typology of ontology nor the construction of layered ontologies. Instead, we focus on the idea of how to *gradually* build ontologies within layered ontologies. In the next subsection, we design the layered structure of DOGMA-MESS using SDT.

3.1 Outline of SDT in the Meaning Evolution Support Systems

Based on the work in [20, 4], we model a scalable ontology into four layers: Meta-Ontology (MO), Upper Common Ontology (UCO), Lower Common Ontology (LCO) and Organizational/Topical Ontology (OTO) (Fig. 1). In [20], *topical ontology structure* for scalable ontology engineering is introduced to represent the knowledge structure of the domain experts (the stakeholders of a community) by involving different view points of analysis and modeling. Later on, the *interorganizational ontology model* is designed to match the requirements for the meaning evolution [4]. We try to integrate SDT into the topical ontology model and interorganizational ontology model as illustrated in Fig. 1.

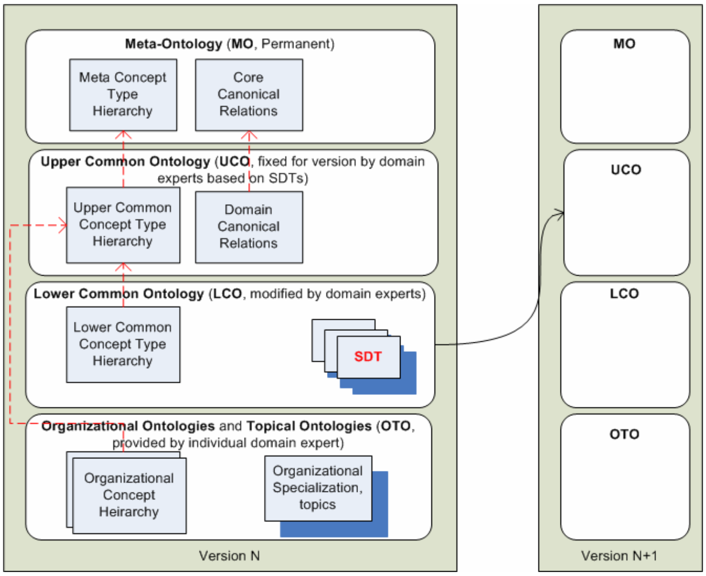

Meta-Ontology (MO, Permanent)
Meta Concept Type Hierarchy
Core Canonical Relations
Upper Common Ontology (UCO, fixed for version by domain experts based on SDTs)
Upper Common Concept Type Hierarchy
Domain Canonical Relations
Lower Common Ontology (LCO, modified by domain experts)
Lower Common Concept Type Hierarchy
SDT
Organizational Ontologies and Topical Ontologies (OTO, provided by individual domain expert)
Organizational Concept Heirarchy
Organizational Specialization, topics
Version N
MO
UCO
LCO
OTO
Version N+1

Fig. 1. Interorganizational Ontology Engineering Model in DOGMA-MESS (with the Integration of SDT). The dotted lines with arrows indicate the specialization dependencies between ontologies of different levels.

An interorganizational ontology evolves and creates different versions over time. The starting point of each version is the current insight about the common interest and common settings, i.e. the services or tasks that an interorganizational ontology is going to be used. The end result of each version is a common ontology, which is a point to reach the *consensus* from various individual *interpretations* of topical or organizational ontologies.

Within a version, four levels of ontologies need to be distinguished:

1) *Meta-Ontology* (MO) defines the abstract concept types, such as ‘Actor’, ‘Object’, ‘Process’ and ‘Quality’. Conceptualization at this level is not allowed to be changed. The relations between these concept types fall into two categories: i) the hierarchical relations reflected by the type hierarchical construct. This kind of relations is also called *subsumption* ontological roles (e.g. “subtype of” relationship in [14]). ii) Other Core Canonical Relations, such as “part-of” *merelogical* relation in [8], “property-of” relation and “equivalent” relation.
2) Each domain has its own *Upper Common Ontology* (UCO); the *Upper Common Concept Type Hierarchy* organizes the (evolving) concept types that are common to the domain. For example, ‘Actor’ at the Meta-Ontology level can be translated into the concepts ‘employer’ and ‘employee’ at the UCO level of the Human Resource Management domain. Domain experts define domain canonical relations in terms of the domain. For example, the domain canonical relation ‘hire’ can be applied between ‘employer’ and ‘employee’. When an ontology evolves, all the concepts in the UCO of *version N* are lifted to the concepts in the UCO of *version N+1*. The *core domain experts* are responsible to standardize the concept definitions at his level.
3) *Lower Common Ontology* (LCO) is the most important and complex layer for the meaning negotiation in DOGMA-MESS. The concept definitions at this level are the organizational and topical specializations, which are created at the Organizational Ontology or Topical Ontology (OTO) level, are aligned and merged. This process happens within a version. The candidate concepts are analyzed by an SDT, which contains the decisions of whether a concept can be lifted (or merged) to the LCO or not. Once the concept is not decided to be lifted, it will keep in the OTO and wait for the next iteration of versioning process. In the next section, a concrete SDT example with a lifting rule is demonstrated.
4) *Organizational Ontology* and *Topical Ontology* (OTO) seek to represent systematically the knowledge structure the domain experts has on given *themes* (or *tasks*) individually. A Topical Ontology “lays foundation for application (or task) specific ontologies and conceptual models… its semantic space covers multiple subjects and dynamic evolution of the core concepts within a topic” [21]. Concepts within a topic represent terminology of application structure, assumption and framework. Within a version, every domain expert (or every enterprise-wise stakeholder group) is responsible to build his own OTO based on the ontology models in UCO. For example, we have a Conceptual Graph model that describes the definition of “Teacher” and “Course” at the UCO level (Fig. 2). Based on it, a domain expert may introduce a new relevant concept “Patience” by adding a conceptual relation “has skill” to “Teacher” (Fig. 3). Similarly, a new relevant concept “Oral comprehension” can be introduced at OTO level (Fig. 4).

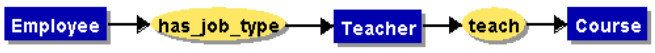


Fig. 2. The concept "Teacher" designed in Conceptual Graph [14] at UCO level

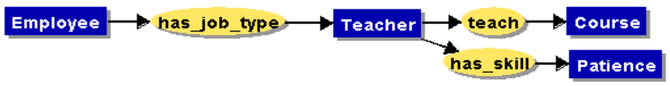


Fig. 3. A new relevant concept "Patience" in Conceptual Graph at OTO level

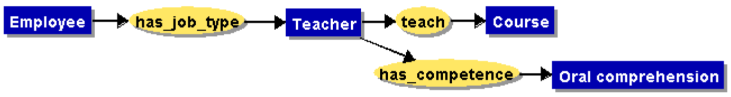


Fig. 4. A new relevant concept "Oral comprehension" in Conceptual Graph at OTO level

When an ontology evolves, many early defined concepts at UCO level may need to be revised. First, these concepts are degraded to OTO level. Then, the domain experts redefine them and put them into the next lifting iteration.

Based on the discussion above, we argue that the lifting rule(s) are the kernel decision rules in DOGMA-MESS. In the next section, we focus on this kind of decision rules and store them as the SDT commitments.

4 Design SDT for DOGMA-MESS

In this section, we try to set up the SDT to assist lifting the concepts from OTO level to UCO level. Different decision rules (the lifting algorithms) are respectively formalized as different sets of SDT commitments (section 4.1 and 4.2). We evaluate these lifting algorithms by analyzing the tabular reports generated by the system (section 4.3).

4.1 Lift a Concept from OTO to LCO

When we lift a concept from OTO level to LCO level, we need to choose some concepts at OTO level. Let $\boldsymbol{S_c}$ be the concept set at OTO level, and let $\boldsymbol{S_l}$ be the resulting lifted concept set at LCO level. In order to compute this process automatically, we hereby introduce two important condition stubs used to form SDT condition lexons – the *relevance score* $\boldsymbol{R_c}$ and the *interest point* $\boldsymbol{I_p}$.

A concept $\boldsymbol{C_i}$ at OTO level is considered as a *relevant* candidate concept when it gets certain amount of *relevance score* $\boldsymbol{R_c}$. R_c is set zero when a new concept is defined at the first time. It increases when the concept is defined in other organizational ontologies designed by different domain experts. For example, if we get the concept "skill of safety" defined within the same context from two different organizational ontologies, the R_c of this concept is increased by one.

Interest point $\boldsymbol{I_p}$ starts from zero. I_p is assigned to an existing concept at UCO level. It increases by one when a new concept, which is connected to this existing concept, is introduced. For example, we have the concept definition of “Teacher” at UCO level (Fig. 2). When a domain expert adds a new relevant concept “Patience” to “Teacher” at OTO level (Fig. 3), I_p of “Teacher” is increased by one. I_p reflects the focusing interests of the stakeholder community. We consider the concept “Patience” as a candidate concept that will be lifted to UCO level only when I_p of its connected concept “Teacher” meets a certain threshold value. When a new concept at OTO level is connected to more than one concept at UCO level, we choose the biggest number of I_p. Accordingly, we formalize a lift rule into the SDT commitments as illustrated in Table 2.

Table 2. An example of SDT commitments and their explanations in the natural language

ID	Commitment	Verbalization
1	(P1 = [concept, has, is of, Ip], P2 = [Ip, is of, has, concept]) : **UNIQ** (P1, P2).	**Each** concept has **at most one** interest point. And **each** interest point is of **at most one** concept.
2	(P3 = [concept , has, is of, Rc], P4 = [Rc, is of, has, concept]) : **UNIQ** (P3, P4).	**Each** concept has **at most one** relevance score. And **each** relevance score is of **at most one** concept.
3	(P5 = [concept, has, is of, Rc], P6 = [concept, has, is of, Ip], P7 = [concept, is lifted to, contain, UCO level]) : **IMP**(**AND**(P5(Rc) **>=** T_1**,** P6(Ip)>=T_2), P7).	The fact that a concept is lifted to UCO level **depends on** two conditions: 1. whether its relevance score is **more than** T_1 or not; **And** 2. Whether its interest point is **more than** T_2 or not.

Commitment 1 in Table 2 uses the uniqueness constraint (‘UNIQ’) to express the one-to-one relationship between ‘concept’ and ‘interest point’. So does commitment 2. Commitment 3 uses the propositional connectives ‘IMP’ (the implication connective) and ‘AND’ (the conjunction connective) to express that: a candidate concept can be lifted to UCO level if and only if its relevance point and interest point meet their threshold (‘T_1’ and ‘T_2’).

Table 3. A tabular report generated based on the SDT, which contains the commitments in Table 2 (T_1=15, T_2=20)

Candidate concept / Condition	Patience	Oral comprehension	…
Super Type	N/A	Competence	…
Relevance Score	30	20	…
Relevant concept at UCO	Teacher	Teacher	…
Interest Point	30	30	…
...	…	…	…
Action/Decision			…
Keep for next MESS iteration	*		
Lift to LCO		*	

Based on Table 2, two concepts at the OTO level – ‘Patience’ and ‘Oral comprehension’, which are considered as two candidate concepts, are analyzed in Table 3. Table 3 contains the decision whether the concepts ‘Patience’[4] and ‘Oral

[4] Its concept is given by Fig. 3.

comprehension'[5] at OTO level can be lifted to LCO level or not. The tabular report is automatically generated by the SDT plug-in in the DOGMA-MESS tool[6]. As the interest point (10) of the concept 'Patience' doesn't reach the threshold (15), it is kept at OTO level for next MESS iterations.

The resulting concept set is then provided to the core domain experts, who are responsible for standardizing the concepts and merging them at UCO level. As the concepts are defined and visualized in the Conceptual Graph models (e.g. Fig. 4), the core domain experts can use many available conceptual graph matching manners, such as in [12], to merge the new concepts automatically into the existing concepts. During this merging phase, some extra links between new concepts and existing concepts need to be constructed. For example, conceptually equivalent concepts need to be linked with the "equivalent" canonical relation. The merging process results in several reorganized conceptual graphs at UCO level.

Table 2 is as an example of a lifting rule. In practice, users are free to choose their preferred lifting rules. In the next subsection, we will improve the lifting rule.

4.2 Improve the Lifting Algorithm

In the PoCeHRMOM[7] project, we observe that not all the parameters used in SDT are equally important. For example, the relevance score R_c is often considered more important than the interest point I_p. In order to emphasize their importance, we use the weights for I_p and R_c. We define a weight set - ω - as a set of real numbers [0, 1].

Let ω_{I_p} be the weight of I_p, ω_{R_c} be the weight of R_c, where $\omega_{I_p}, \omega_{R_c} \in \omega$.

Let T be a *threshold* value, c_i be a concept and S_l the resulting concept set. If c_i fulfills the following condition:

$$\omega_{I_p} * I_p + \omega_{R_c} * R_c \geq T \tag{1}$$

Then, we say that $c_i \in S_l$.

As discussed in section 3.1, the concepts at the OTO level are developed within *different* topics. We argue that the choice of a topic is not neutral. A topic reflects the stakeholders' motivations and the purpose of using an ontology, although it abstracts away the application perspectives. Let C' be a topic. We use the notion $C'.c_i$ to indicate that c_i is defined within C'. A concept c_i can have several prefix C' as its topics can be layered. For example, the concept 'oral comprehension' can be defined in both the topics of 'the profile for a teacher' and 'the profile for a marketing manager'. In this case, we use the notion $C'_1.C'_2...C'_n.c_i$.

In practice, we observe that the more topic prefixes are used to define a concept, the more re-useful the concept becomes. Therefore, we use the number of the topic prefixes of a candidate concept as the third criteria. Let N_c be the number of the topic prefixes of c_i and ω_{topic} be the weight to adjust N_c, where $\omega_{topic} \in \omega$.

[5] Its concept is introduced by Fig. 4.

[6] The DOGMA-MESS tool currently developed in STARLab is a web portal to assist domain experts to design ontologies: http://www.dogma-mess.org/

[7] PoCeHRMOM project aims to establish semantic rich knowledge of human resource management by means of ontology and Semantic Web technologies.

Now the decision rule is modified into –
If c_i fulfills the following condition:

$$\left(\omega_{Ip} * I_p + \omega_{Rc} * R_c + \omega_{topic} * N_c\right) \geq T \tag{2}$$

Then, we say that $c_i \in S_l$.

Table 4. The optimized lifting rule formalized as a SDT commitment.

(P5 = [concept, has, is of, Rc],
P6 = [concept, has, is of, Ip],
P7 = [concept, has, is of, Nc],
P8 = [concept, is lifted to, contain, UCO level])
: **IMP**((P5(Rc)*Wrc + P6(Ip)*Wip + P7(Nc)*Wtopic) >= T, P8).

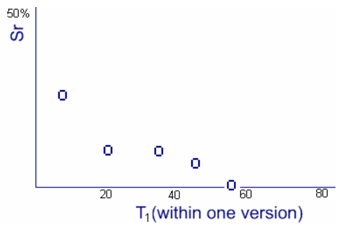
Table 4 illustrates the SDT commitment for the improved decision rule. In the next subsection, different decision rules are evaluated.

4.3 Evaluation of Different Decision Rules

We evaluate our lifting algorithms illustrated in section 4.1 and section 4.2 with the selection rates. Let $\boldsymbol{N_{can}}$ be the number of the candidate concepts, which resides at OTO level. And let $\boldsymbol{N_{sel}}$ be the number of selected concepts that are lifted. The concept selection rate S_r is defined as:

$$S_r = \frac{N_{sel}}{N_{can}}$$

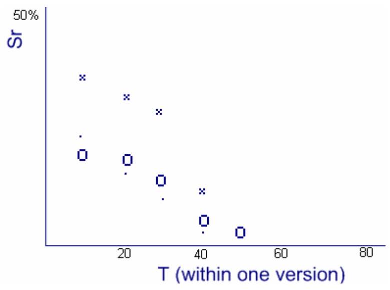
Concerning the decision rule in section 4.1, S_r is studied by setting up different values of T_1 and T_2.

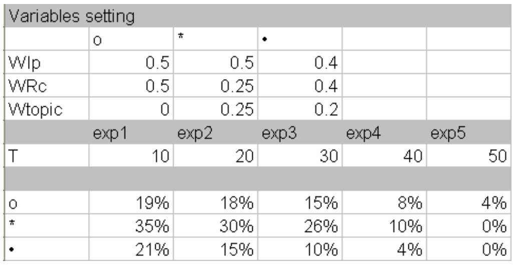


	exp1	exp2	exp3	exp4	exp5
T1	8	20	37	44	56
T2	15	15	15	15	15
o	25%	12%	14%	10%	0%

Fig. 5. Selection rates and their corresponding values of the parameters when applying the algorithm in section 4.1

Fig. 5 illustrates S_r by applying the first algorithm (section 4.1). The selection rates are initially calculated when we provide different threshold values T_1 and T_2. We make two sets of experiments: First, the value of T_1 is fixed and the value of T_2 is changeable; second, the value of T_2 is fixed and the value of T_1 increases every time. Both experiments are performed five times. Fig. 5 is an example that takes the fixed values of T_2 and the dynamic values of T_1. We observe that the value S_r gets smaller when the threshold values T_1 and T_2 increase. At a certain moment, no concept can be lifted as the threshold values are set too big. In practice, the values of T_1 and T_2 are adjusted based on the real situations. For instance, the core domain experts raise T_1 and T_2 when there are two many candidate concepts and they don't have a lot of time.





Variables setting					
	o	*	•		
WIp	0.5	0.5	0.4		
WRc	0.5	0.25	0.4		
Wtopic	0	0.25	0.2		
	exp1	exp2	exp3	exp4	exp5
T	10	20	30	40	50
o	19%	18%	15%	8%	4%
*	35%	30%	26%	10%	0%
•	21%	15%	10%	4%	0%

Fig. 6. Selection rates and their corresponding values of the parameters when applying the algorithm in section 4.2

Fig. 6 illustrates the selection rates by applying the algorithm in section 4.3. We experiment on three value sets of the parameters W_{Ip}, W_{Rc} and W_{topic}. The balance between these parameters reflects the preference of the core domain expert. As the previous algorithm, the selection rate decreases when the threshold value is big. No concept can be lifted when the threshold value meets the limitation.

Comparing these two sets of decision rules, we discuss that the first one is simple and easy understandable. The second one is more complicated; however, it is more useful and precise as it reflects the preference of the key domain experts. The two algorithms shown in this paper are two examples. In practice, more decision rules may be introduced by the key domain experts. For example, the parameter of conceptual similarity may be added to the candidate concepts.

5 Related Work and Discussion

Nowadays, current ontology merging, versioning and alignment methods mainly focus on how to *integrate* different ontologies, such as in [10]. Researches concentrate on combining several (sub-) ontologies into one ontology by removing inconsistency and reducing conflicts among them. DOGMA-MESS does not focus on how to solve these problems, but to *gradually* build *interoperable* ontologies amongst a large, knowledge-intensive community. One *communicates* with others' needs, trying to find the overlapping interests, with which we make *interorganizational ontologies*.

Consensual knowledge base introduced in [6] and *cooperative domain ontology* studied in [1] are promising related work in building consensus on ontology level. However, authors in [4] discuss that those methodologies are lack of community consideration although those methodologies work out some basic principles for building ontological consensus.

In the Prolix project, we observe that there are several advantages and disadvantages while applying SDT to DOGMA-MESS. By using SDT in DOGMA-MESS, the effectiveness of DOGMA-MESS processes increases. Before, we used to hardcode the algorithms in the system. Now, the core domain experts only need to create the SDT commitments while they want to apply new decision rules. During the exercises, we observe that the tabular reports generated based on SDT's are extremely convenient and user-friendly for non-technical domain experts. Another advantage is the flexibility. DOGMA-MESS using SDT is more flexible because the specifications and decision rules are not hard coded in the system any more. The knowledge engineers can create different algorithms and decision rules based on their needs.

A big disadvantage is the complexity. The knowledge engineers need to know how to write the SDT commitments.

6 Conclusion and Future Work

In this paper, we focus on the discussion on the role of SDT (Semantic Decision Table) in Meaning Evolution Support Systems for ontology engineering. SDT is used to improve the effectiveness of such systems for a better design. In this paper, SDT are explicitly used when new relevant concept need to be defined and lifted to the domain level. The decision rules, with which the system decides to lift a certain number concepts, are formalized as SDT commitments. We evaluate different decision rules by analyzing the relations between some specific parameters of the decision rules (the input) and the selection rates of concepts (the output).

We have developed a tool to support constructing SDT. The current version supports some specific commitment types [16]. A web portal to support DOGMA-MESS methodology has been developed [4]. A future work is to integrate SDT modules to DOGMA-MESS methodology, and reason the SDT commitments.

Acknowledgments. The experimental data is collected from the IWT PoCeHRMOM project. The research is partly supported by the EC Prolix project (FP6-IST-027905) and the EC AmIE project (ITEA 2 - 06002). It's author's pleasure to thank Pieter De Leenheer for the paper discussion.

References

[1] Aschoff, F.R., Schmalhofer, F., van Elst, L.: Knowledge mediation: a procedure for the cooperative construction of domain ontologies. In: proc. of the ECAI 2004 workshop on Agent-Mediated Knowledge Management, pp. 29–38 (2004)

[2] CSA, Z243.1-1970 for Decision Tables, Canadian Standards Association (1970)

[3] de Moor, A.: Ontology-Guided Meaning Negotiation in Communities of Practice. In: Mambrey, P., Gräther, W. (eds.) C&T 2005. Proc. of the Workshop on the Design for Large-Scale Digital Communities at the 2nd International Conference on Communities and Technologies, Milan, Italy (2005)

[4] de Moor, A., De Leenheer, P., Meersman, R.: DOGMA-MESS: A Meaning Evolution Support System for Interorganizational Ontology Engineering. In: Schärfe, H., Hitzler, P., Øhrstrøm, P. (eds.) ICCS 2006. LNCS (LNAI), vol. 4068, pp. 189–203. Springer, Heidelberg (2006)

[5] De Leenheer, P., de Moor, A., Meersman, R.: Context Dependency Management in Ontology Engineering. In: Spaccapietra, S., Atzeni, P., Fages, F., Hacid, M.-S., Kifer, M., Mylopoulos, J., Pernici, B., Shvaiko, P., Trujillo, J., Zaihrayeu, I. (eds.) Journal on Data Semantics VIII. LNCS, vol. 4380, pp. 26–56. Springer, Heidelberg (2007)

[6] Euzenat, J.: Building consensual knowledge bases: context and architecture. In: Mars, N.J.I. (ed.) Proc. of the KB&KS 1995. Towards Very Large Knowledge Bases, pp. 143–155. IOS Press, Amsterdam (1995)

[7] Gruber, T.R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In: Guarino, N., Poli, R. (eds.) Workshop on Formal Ontology, book Formal Ontology in Conceptual Analysis and Knowledge Representation, Padva, Italy, Kluwer Academic Publishers, Dordrecht (1993)

[8] Guarino, N., Poli, R. (eds.): Formal Ontology in Conceptual Analysis and Knowledge Representation. Special issue of the International Journal of Human and Computer Studies 43(5/6) (1995)

[9] Halpin, T.: Information Modeling and Relational Database: from Conceptual Analysis to Logical Design. Morgan-Kaufmann, San Francisco (2001)

[10] Madhavan, J., Bernstein, P., Domingos, P., Halevy, A.: Representing and reasoning about mappings between domain models. In: AAAI 2002. Eighteenth National Conference on Artificial Intelligence, Edmonton, Canada, pp. 80–86. American Association for Artificial Intelligence (2002). ISBN:0-262-51129-0

[11] Meersman, R.: The Use of Lexicons and Other Computer-Linguistic Tools in Semantics, Design and Cooperation of Database Systems. In: CODAS 1999. The Proceedings of the Second International Symposium on Cooperative Database Systems for Advanced Applications, pp. 1–14. Springer, Heidelberg (1999)

[12] Myaeng, S.H., Lopez-Lopez, A.: Conceptual graph matching: a flexible algorithm and experiments. International Journal of Pattern Recognition and Artificial Intelligence 4, 107–126 (1992)

[13] Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Massachusetts (1984)

[14] Sowa, J.: Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks Cole Publishing, Pacific Grove (2000)

[15] Spyns, P., Meersman, R., Jarrar, M.: Data Modeling versus Ontology Engineering. SIGMOD Record: Special Issue on Semantic Web and Data Management 31(4), 12–17 (2002)

[16] Tang, Y., Meersman, R.: Towards Building Semantic Decision Table with Domain Ontologies. In: Man-Chung, C., Liu, N.K., Cheung, J., Zhou, R. (eds.) ICITM 2007. proceedings of International Conference on Information Technology and Management, pp. 14–22. ISM Press, Hong Kong (2007)

[17] Tang, Y., Meersman, R.: On Constructing Semantic Decision Tables. In: Wagner, R., Revell, N., Pernul, G. (eds.) DEXA 2007. proc. of 18th International Conference on Database and Expert Systems Applications. LNCS, vol. 4653, pp. 34–44. Springer, Heidelberg (2007)

[18] Tang, Y.: On Conducting a Decision Group to Construct Semantic Decision Tables. In: OntoContent workshop, in proc. of OTM 2007 (this book)

[19] Tang, Y.: A Theoretic Foundation of Semantic Decision Tables and Decision Groups, PhD report, VUB STARLab, 2007.

[20] Zhao, G., Meersman, R.: Architecting ontology for Scalability and Versatility. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. LNCS, vol. 4276, pp. 1605–1614. Springer, Heidelberg (2006)

[21] Zhao, G., Meersman, R.: Towards a Topical Ontology of Fraud. In: Mizoguchi, R., Shi, Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 566–572. Springer, Heidelberg (2006)

Extending Online Travel Agency with Adaptive Reservations

Yu Zhang[1], Wenfei Fan[2], Huajun Chen[1], Hao Sheng[1], and Zhaohui Wu[1]

[1] College of Computer Science,
Zhejiang University, Hangzhou 310027, Zhejiang, China
{yzh,huajunsir,wzh}@zju.edu.cn
[2] University of Edinburgh & Bell Laboratories
wenfei@inf.ed.ac.uk

Abstract. Current online ticket booking systems either do not allow customers to reserve a ticket with a locked price, or grant a fixed reservation timespan, typically 24 hours. The former often leads to *false availability*: when a customer decides to purchase a ticket after a few queries, she finds that either the ticket is no longer available or the price has hiked up. The latter, on the other hand, may result in unnecessary *holdback*: a customer cannot purchase a ticket because someone else is holding it, who then cancels the reservation after an excessively long period of time. False availability and holdback routinely lead to loss of revenues, credibility and above all, customers. To rectify these problems, this paper introduces a transaction model for e-ticket systems to support a reservation functionality: customers can reserve tickets with a locked price, for a timespan that is determined by the demands on the tickets, rather than being fixed for all kinds of the tickets. We propose a method for implementing the model, based on hypothetical queries and triggers. We also show how to adjust the reservation timespan *w.r.t.* demands. We experimentally verify that our model and methods effectively reduce both false availability and holdback rates. These yield a practical approach to improving not only e-ticket systems but also other e-commerce systems.

1 Introduction

It is increasingly common for people to book travel packages online. A number of online ticket booking systems (*a.k.a.* e-ticket systems or virtual travel agents) have been launched by airlines [1] or Web service providers (*e.g.,* Expedia [2], Orbitz [3] and Priceline [4]). While these services allow customers to query airfare and purchase tickets online, they provide very limited support for one to *reserve* a ticket with a locked price.

It is evident that customers want to *reserve* a ticket before they are ready to purchase it. For instance, Alice wants to book a ticket via an online service. After finding a ticket with a reasonable price, she proceeds to build up the rest of her travel plan by, *e.g.,* issuing queries about hotels and car rental. After 5 minutes, she is happy with the package she found and decides to purchase the ticket. But she is frustrated to find that either the ticket she liked has already been sold out, or the price has gone up. She has to start the process again from scratch, and may get her travel package in place only after repeated failures. She would certainly like it if the system had allowed her to reserve the ticket with the price locked *when* she found it.

Existing e-ticket systems support reservations based on one of the following approaches.

(a) On one end of the spectrum, reservations are treated as "purchases": as soon as tickets are reserved, they are blocked from other customers *until* they are finally purchased or the reservation is canceled. The other customers cannot view or query these tickets as if they were already sold. However, typically certain percentage of the reservations will be canceled in the end. As a result, this *conservative* approach inevitably incurs excessive and unnecessary rollbacks and transaction management cost. Worse still, with this comes *holdback* of tickets: other customers are denied the chance to purchase those tickets that are reserved but finally *not* purchased. This often leads to loss of revenue, among other things.

(b) On the other end of the spectrum, reservations are treated as "queries": no reservation action is taken. This *aggressive* approach leads to *false availability*: a customer finds tickets with a reasonable price; however, when she is ready to commit to purchase, she is told that the tickets with the price are actually not available. While this approach does not suffer from the holdback problem, it leads to loss of credibility and eventually loss of customers.

(c) Compromising these two extreme approaches, some services allow one to reserve a ticket for a *fixed* period of time, typically 24 hours. That is, a reservation will expire after 24 hours if no explicit purchase or cancellation is conducted. This reduces holdback and false availability rates, but only to an extent: in a travel season when certain tickets are highly demanded, a fixed 24-hour reservation timespan is often excessively long and may incur the same holdback problem as approach (a) above. In practice the reservation timespan should be *adaptive*, *i.e.,* it should *vary* in accordance to the demands on the tickets. Furthermore, it is common that customers decide to commit to purchase in a single session of querying and purchasing; thus a fixed 24-hour timespan is often an overkill.

We have investigated 30 popular e-ticket systems, and found that all except seven adopt approach (b), *i.e.,* no reservation function is supported at all. The seven services that support reservation all follow approach (c) (American Airline, Continental, Southwest, United, and US Airways [1], MakemyTrip.com and Cfares.com): customers are allowed to hold the tickets for 24 hours or until the midnight of the next day. In addition to the holdback problem with approach (c), these services do *not* allow the price of a reserved ticket to be locked; thus although a customer can reserve a ticket she is not guaranteed to get the ticket with the price she found when making the reservation. In short, the reservation functionality supported by existing online ticket booking systems neither rectifies the holdback problem nor reduces false availability.

Contributions. To this end we propose a transaction model for e-ticket systems to support the reservation functionality while reducing both the holdback rate and the false availability rate. In this model the customer is allowed to reserve a ticket, with a *locked* price, for a timespan adjusted in accordance to demands on the tickets. We also provide an efficient technique to support the model such that the addition of the reservation function does not imply any drastic degradation in performance for existing e-ticket systems.

The main contributions of the paper include the following:

1. A transaction model that supports queries, purchases and in addition, *reservations.* It reduces the holdback and false availability rates by (a) adapting reservation timespan based on demands, and (b) making a percentage of reserved tickets available to other customers, determined by an estimate of the reservations that eventually turn into purchases.
2. A combination of techniques, including hypothetical queries (see, *e.g.,* [5,6]) and triggers (see, *e.g.,* [7,8]), to efficiently support the transaction model.
3. A method for computing the reservation timespan based on demands on different tickets.
4. A preliminary experimental study that verifies the effectiveness and efficiency of our techniques.

These will provide online travel systems with a practical method to support the reservation functionality. We should remark that although we focus on ticket booking systems to simplify the discussion, the techniques proposed in this work are generic enough to be applicable to other e-commerce systems such as finance services.

Organization. We introduce our transaction model and implementation technique, as well as a method for computing reservation timespan in Section 2. Our experimental results are presented in Section 3, followed by related work and future work in Section 4.

2 A Transaction Model

In this section we first present the transaction model. We then present techniques for efficiently implementing the model. Finally, we outline the architecture of an e-ticket system based on this model.

2.1 Supporting Reservations

We propose an e-ticket system that allows customers to reserve tickets, in addition to querying and purchasing. A customer may reserve tickets with a locked price, for a timespan s computed by the system based on demands on the tickets. From the time when the reservation is granted until the end of the timespan s, the customer may either (a) commit to purchase the tickets or (b) cancel the reservation. A short while before the end of s, the system sends a reminder to

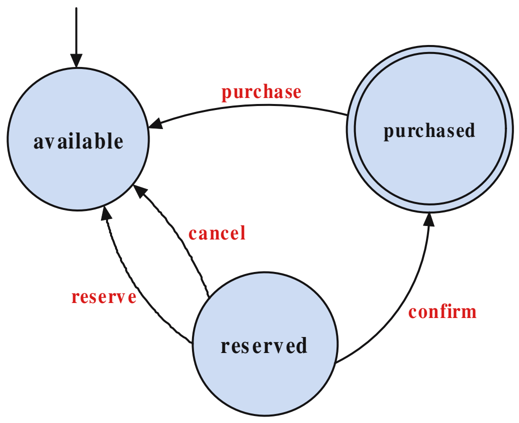


Fig. 1. FSM representation of transactions of e-Ticket System

the customer about the reservation; if no action is taken by the customer, the system cancels the reservation at the end of s, *i.e.,* when the reservation *expires*.

In the transaction model for the e-ticket system, each ticket is associated with one of following *states*: available, reserved or purchased. The state of a ticket changes from available to reserved when the ticket is reserved by a customer. It in turn changes from reserved to purchased if the customer commits to purchase, and to available if the reservation is canceled either by the customer or by the system when the reservation expires. While transition may take place from available to purchased, it cannot change from purchased to available or reserved. More formally, this can be characterized in terms of a (deterministic) finite state machine (FSM), which is commonly used in modeling Web services (see, *e.g.,* [9,10]).

FSM=(Σ,S,s_0,Δ,F), where:

- $\Sigma = \{reserve, cancel, confirm, purchase\}$
- $S = \{\text{available}, \text{reserved}, \text{purchased}\}$
- $s_0 = \{\text{available}\}$
- $\Delta = S \times \Sigma \rightarrow S$
- $F = \{\text{purchased}\}$

Figure 1 shows an abstract representation as FSM of the reservation process. We refer to the set τ of tickets of each flight (train, car, etc) as tickets of *type* τ.

- We say that tickets of type τ are *held-back* if when a customer wants to purchase or reserve a ticket of type τ, all tickets of τ are either purchased or reserved at the moment, and moreover, some of those reserved tickets change to available later on.
- We say that tickets of τ are *falsely available* if a customer is allowed to reserve a ticket of τ but later on cannot purchase it (*i.e.,* change its state from reserved to purchased).

Let N denote the number of successful reservations made on tickets of τ that eventually change from reserved to purchased, n_h the number of failed reservations when tickets of τ are held-back, and n_f the number of reservations on tickets that are falsely available. We define the false availability rate and hold-back rate to be $n_f/(N+n_f)$ and $n_h/(N+n_h)$, denoted by δ and γ, respectively.

We aim to reduce both δ and γ. Since it is not always possible to minimize both the false availability rate and the holdback rate, we give higher priority to reducing the false availability rate since it typically inflicts more severe damages when credibility and customers are concerned. Below we propose two methods to reduce false availability rate δ while keeping the the holdback rate γ low.

First, we introduce a parameter α in the range $[0, 1]$, referred to as the *purchase rate*, for the set of tickets of each type τ. The purchase rate indicates (an estimate of) the percentage of the tickets of which the states change from reserved to purchased. Intuitively, we make $(1 - \alpha)$ percent of reserved tickets available to customers, so that not all "hot" tickets would be held back. Observe that the smaller α is, the less the holdback rate is, but on the other hand, the higher the false availability rate is.

Second, we assign different reservation timespan s to different types of tickets. In Section 2.2, we shall present a method to compute s based on demands on tickets of type τ. Intuitively, the smaller s is, the less the holdback rate is. When it comes to the false availability rate δ, the story is a bit more complicated. Making s larger may on one hand hold the ticket longer so that when the customer decides to commit to the purchase, the ticket will still be available; but on the other hand, if the purchase rate α is small, the chances are that the reserved tickets become unavailable and thus it may leads to higher δ. As will be seen in Section 3, making s larger may reduce δ only if α is sufficiently large.

For the reservation approaches adopted by existing e-ticket systems surveyed in Section 1, approach (b) adopts $\alpha = 0$ and s does not exist, while approaches (c) uses $\alpha = 1$ and fixes s to be 24 hours. In our model, we set α to a value between 0 and 1, determined by statistical analysis and estimate, and will be seen in Section 2.2, we compute s based on demands on tickets of type τ rather than giving a fixed timespan for all types of tickets. As will be seen in Section 3, we keep α and s sufficiently large so that the false availability rate remains low while the holdback rate is reduced.

2.2 Implementation Techniques

A naive approach to implementing reservations is to create and maintain relations for storing information about available, reserved and purchased tickets, and, whenever the state of a ticket changes from reserved to purchased (resp. available), or the other way around, we modify both the reserved and the purchased (resp. available) tables. This, however, incurs heavy transaction cost. In light of this, we propose a technique based on hypothetical queries to reduce the overhead. We also present a method to compute the reservation timespan based on demands on different tickets.

Relations for Reserved, Purchased and Available Tickets. Along the same lines as existing e-ticket systems, we store information about tickets of various states in *fact* tables:

- Ticket table $T(\mathsf{id}, \sharp, \mathsf{price})$ keeps track of the number $\sharp$ of available tickets of a specific flight id.
- Reservation table R $(\mathsf{id}, \sharp, \mathsf{t_s}, \mathsf{price}, \mathsf{info})$ stores reservations made so far. A customer reserves $\sharp$ tickets on flight id with a locked price, where t_s is a timestamp specifying when the reservation expires, and info denotes some basic information about the customer such as name, nationality, number of tickets, etc.
- Purchase table $P(\mathsf{id}, \sharp, \mathsf{price}, \mathsf{info})$ stores the *real* purchases.
- Other tables store other information about the flight such as departure time, arrival time and destinations, etc.

Hypothetical Queries. A *hypothetical query* is of the form Q when $\{\{U\}\}$, where U is an update (see [6]). It to find the value that query Q would return on a database DB that would be obtained by executing update U on the original DB, *without* actually updating DB.

We regard a reservation as a *hypothetical purchase* while final commitment to purchase as a *real purchase.* Capitalizing on hypothetical queries, when a reservation is made, we *only* need to modify the reservation table R, *without* changing either T or P. We modify T, R and P only when a reservation is converted into a real purchase. This reduces the overhead of unnecessary transactions and rollbacks.

To carry this out, for each query Q on T, we automatically rewrite it into a hypothetical query $Q_T = Q$ when $\{\{U\}\}$ on both T and R, such that α percent of the reservations in R are "taken out" from T during the process, where α is the purchase rate given earlier. More specifically, the update U is of the form:

$$U ::= del(T, \alpha R) \quad \text{delete } \alpha \cdot \sharp \text{ tickets of } R \text{ from } T$$

In a nutshell, for each occurrence of T in Q, we replace it with $del(T, \alpha R)$ by taking out certain tickets already reserved. Thus

$$Q_T = Q \text{ when } \{\{del(T, \alpha R)\}\} \tag{1}$$

As remarked earlier, we make $(1 - \alpha)$ tickets in R available to customers to reduce the holdback rate, since typically only α percent of reservations will lead to real purchases in the end.

A number of techniques have been developed for efficiently evaluating hypothetical queries. Here we adopt the lazy approach of [6]. More specifically, we first rewrite Q_T into an equivalent, non-hypothetical query $Q_T^{'}$ by transforming each U into an "explicit substitution", and then applying the substitution and obtaining a pure SQL query. We illustrate this by using a hypothetical query of form (1), where Q is an SQL query for finding the number of tickets in stock for flight k with price p, *i.e.,*

$$Q = \pi_{\sharp}(\sigma_{\mathsf{id}=k \wedge \mathsf{price}=p}\, T). \tag{2}$$

We replace the update with explicit substitution:

$$\pi_\sharp(\sigma_{\mathsf{id}=k \wedge \mathsf{price}=p}\, T) \text{ when } \{\{(T - \alpha R)/T\}\}$$

Note that R is a bag of records since for each flight there may be several reservations for it by different customers. Thus to deduct the total number of reserved tickets in R, we need to use aggregate function sum. Now we apply the substitution to the query Q_T and get

$$Q_T^{'} \equiv \pi_\sharp(\sigma_{\mathsf{id}=k \wedge \mathsf{price}=p}\, T) - \alpha\, \mathsf{sum}(\pi_\sharp(\sigma_{\mathsf{id}=k \wedge \mathsf{price}=p} R))$$

This query is equivalent to Q_T. Similarly we can automatically rewrite other queries, *e.g.,* queries for finding airfare.

The use of hypothetical queries and automated query rewriting allows us not to update T and P when making or canceling a reservation, and thus reduce the overhead of transactions.

Making, Canceling and Committing Reservations. As remarked earlier, there is no need to update the ticket and purchase tables T and P when a reservation is made or canaled. Indeed, only the reservation table R needs to be changed in response to these updates. In contrast, all three tables T, R and P are updated when a user commits to purchase a reserved ticket.

Making a reservation. Upon receiving a customer request for reserving n tickets for flight k with a locked price p, the system does the following, in *one transaction.* (i) It generates a query Q of form (2) given above, which is to find the number of available tickets for flight k with price p. The query Q is rewritten into a *hypothetical query* of form (1) above, and is evaluated on the ticket table T and the reservation table R using the evaluation techniques described above. If the result of the query Q is negative, then the customer request is *denied.* Otherwise the following steps are taken. (ii) It computes the reservation timespan s and based on s, timestamp $t_s = t + s$, where t is the current time. We will show how s is computed shortly. (iii) It inserts a tuple $(\mathsf{id} = k, \sharp = n, \mathsf{price} = p, \mathsf{info} = i)$ into the reservation table R, where i is the related customer information. Note that neither table T nor table P is updated.

Canceling a reservation. When a customer requests to cancel a reservation of n tickets for flight k, the systems finds the corresponding tuple from the reservation table R, based on both the flight and customer information. It then removes the tuple from R. No other tables are updated.

Committing to purchase a reserved ticket. When a customer commits to purchase reserved tickets, the system does the following, in *one* transaction. (i) It first identifies the corresponding tuple $r = (\mathsf{id} = k, \sharp = n, \mathsf{price} = p, \mathsf{info} = i)$ from the reservation table R, and removes r from R. (ii) It then forms and evaluates query Q as in step (i) for making reservations. If the result of Q is negative, then the tickets are *falsely available* and the transaction aborts. Otherwise the system proceeds to do the following. (iii) It inserts r into purchase table P.

(*iv*) It also updates ticket table T by removing n tickets of flight k from T. Upon the completion of the transaction, the hypothetical purchase of this reservation becomes a *real* purchase. Note that only at this stage all three tables T, R and P need to be modified.

A subtle issue arises when the price of tickets for flight k in the ticket table T is updated. To keep the price of the reserved tickets for flight k unchanged, the system does the following. (*i*) It first finds the total number n of reserved tickets for flight k in the reservation table. (*ii*) In the ticket table T, it keeps the price of $\alpha \cdot n$ tickets for flight k unchanged, while updating the price of the remaining tickets for flight k, where α is the purchase rate.

Adapting Reservation Timespan. As remarked earlier, we determine the reservation timespan for different tickets based on demands on the tickets. More specifically, for tickets of each type τ, we characterize the demands on the tickets using the following parameters.

– *weight* w in the range $[0, 1]$, indicating the "popularity" of the flight; this is determined by statistical analysis of the historical data of the flight, and may vary in different travel seasons; indeed, the demand for flights to Orlando is typically higher before Christmas than that during school terms;
– *advance parameter* d, which is the number d of days prior to the departure of the flight when the reservation is made;
– the *maximum timespan* s_{max}; here one may use 24 hours as the default value, following the practice of most airlines.

We use the following simple formula to compute the reservation timespan s based on these parameters:

$$s = (1 - w) \cdot f(d) \cdot s_{\mathsf{max}}, \tag{3}$$

where $f(d)$ is a function in which c is a constant in $[0, 1]$:

$$f(d) = \begin{cases} c \cdot d/14 & \text{if } d \leq 14 \\ c & \text{otherwise} \end{cases}$$

Intuitively, the more popular the flight is, the less reservation timespan s is; furthermore, the less days in advance the reservation is made, the less s is. In function $f(d)$ we choose constant 14 in accordance to the common practice of most airlines: "penalty" is incurred if the reservation is made within two weeks prior to the scheduled departure of the flight.

Triggers for Expiring Reservations. Shortly before a reservation expires, the system sends the customer a reminder. Recall that when a customer reserves a ticket, a reservation timespan s and a timestamp t_s are computed and stored in the corresponding tuple in the reservation table. That is, the reservation remains valid for a period s of time until time t_s. Meanwhile, a *trigger* is set up such that shortly before the reservation expires, say 30 minutes before t_s, action will be triggered to generate the reminder and notify the customer. If the customer takes no action before the reservation expires, the system performs the cancellation operation at time t_s, as described above.

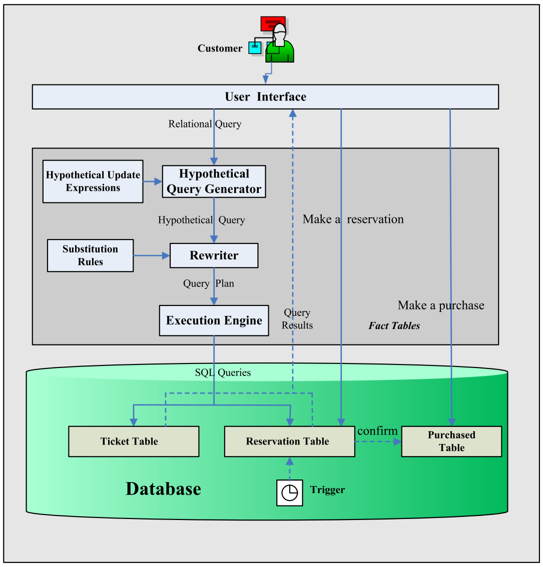


Fig. 2. The Architecture of an e-Ticket System

2.3 The Architecture of a Ticket Booking System

Putting these together, we propose to develop a 3-tier ticket booking system based on our transaction model, as depicted in Figure 2. The top layer is the user interface for customers to book tickets, the bottom layer is the underlying database storing *fact* tables, and the middle-tier processes customer queries, reservation requests and purchase orders. More specifically, upon receiving a customer query, the middle-tier converts it into an equivalent hypothetical query and evaluates it following the strategy given in Section 2.2. Upon receiving a request for making a reservation, the middle-tier starts a transaction to process it as described in Section 2.2. In particular, it computes the reservation timespan using the formula given in Section 2.2. The customer may succeed in making the reservation if there are enough tickets available. Subsequently the customer may cancel the reservation or commit to purchase the reserved tickets, which are again processed using the strategies given in Section 2.2. Furthermore, when a reservation is made, a trigger is set up, such that the customer will be notified shortly before the reservation expires, as described in Section 2.2.

3 Experimental Study

Our experimental evaluation focuses on the effectiveness of our reservation model in reducing the holdback and false availability rates δ and γ. We compare our approach against the approaches adopted by existing e-ticket systems, namely, approaches (b) and (c) described in Section 1, referred to as the *no-hold* and *fully-hold* approaches, respectively. We also investigate the impact of the purchase parameter α and reservation timespan s on δ and γ.

3.1 Experimental Setting

We considered a flight with up to 350 seats initially available. We randomly generated a set of reservations and subsequent confirmations or cancellations. Each reservation requested n tickets, where n is a number randomly chosen from [1,5]. We assumed that the flight was popular: there were sufficiently many customers to query about and book tickets until all the tickets would be sold out. In a duration of 30 days, we assumed that the arrival of customers followed a Poisson process (see, *e.g.,* [11]) and customer arrival rate is set to $\lambda = 1$ unless other specified. The intervals between successive customer arrivals were treated as independent random variables. The latency between a reservation and its subsequent confirmation or cancellation was also generated randomly within time slot $[0, s]$; that is, we assumed that most of customers will inspect their reservations within the timespan s offered by the e-ticket system. For each reservation, we generated another random value to determine whether or not the reservation is confirmed or canceled. Unless specified otherwise, we fixed the confirmation rate to 70% and customer arrival rate to $\lambda = 1$.

The experiments were run on a machine with a 2.40GHZ Pentium IV processor and 512MB of RAM. Each experiment was run 200 times and the average is reported here.

3.2 Experimental Results

we study the impact of the purchase rate α and reservation timespan s on reducing the false availability and holdback rates δ and γ. More specifically, with s (resp. α) fixed to certain values we investigate the effect of varying α (resp. s). The goal is twofold: first, we want to verify that the use of α and s indeed reduces δ and γ; second, we want to study the behavior of different s and α, and find out what values we should choose for them. For the reservation timespan s we either use fixed values or vary its values from 0 to 24 hours, rather than using formula (3) given in Section 2.2.

Recall notations N, n_f and n_h from Section 2.1. In each run, N records the total number of reservations that lead to purchases, n_h the total number of held-back reservations and n_f the total number of reservations on falsely available tickets (*i.e.,* the reservations that will not turn into real purchases when customers come back and inspect their reserved tickets, before or after the reservations expire), in the *entire* life-cycle of ticket selling of the flight. The average of 200 runs is reported.

The effect of varying the purchase rate α. Fixing reservation timespan s to be 6, 12 and 24 hours, we vary the purchase rate α and study its effect on the false availability and holdback rates δ and γ. Observe that when $\alpha = 0$, it characterizes the *no-hold* approach, and on the other hand, when $\alpha = 1$, it corresponds to the *fully-hold* approach. As Figures 3 and 4 show, as expected,

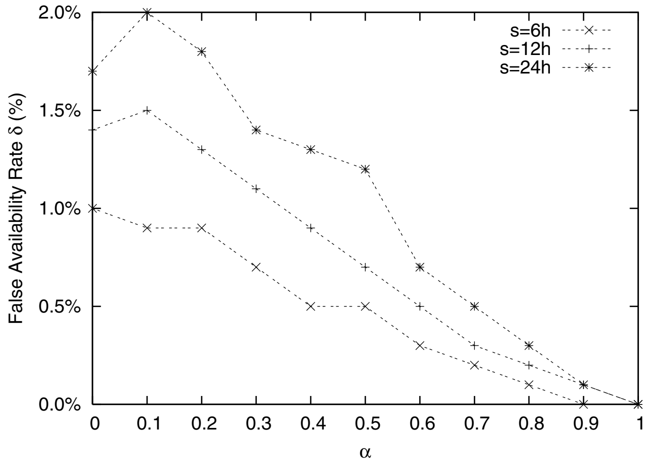


Fig. 3. Effect of varying α on the false availability rate($s = 5, 12, 24$ hours)

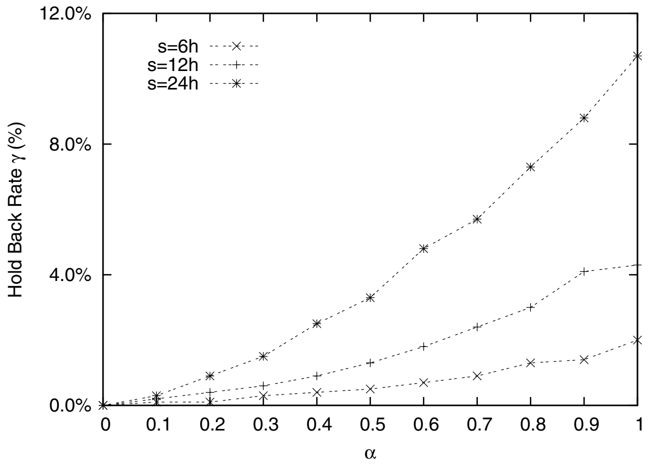


Fig. 4. Effect of varying α on the holdback rate ($s = 5, 12, 24$ hours)

when α increases the holdback rate increases while the false availability rate decreases. It also tells us that while the *no-hold* approach does not have the holdback problem, it incurs a rather high false availability rate. On the other hand, while the *fully-hold* approach does not lead to false available reservations, its holdback rate is rather high. In contrast, if we choose $\alpha \geq 0.8$, the false availability rates reduce to a neglectable value when either $s = 6$, $s = 12$ or $s = 24$, while their holdback rates are lower than the *fully-hold* counterparts.

The effect of varying the reservation timespan s**.** Fixing α to be $0, 0.5, 0.8$ and 1, we vary s from 0 to 24 hours and measure the false availability and holdback rates δ and γ for each s. As shown in Figures 5 and 6, when α increases,

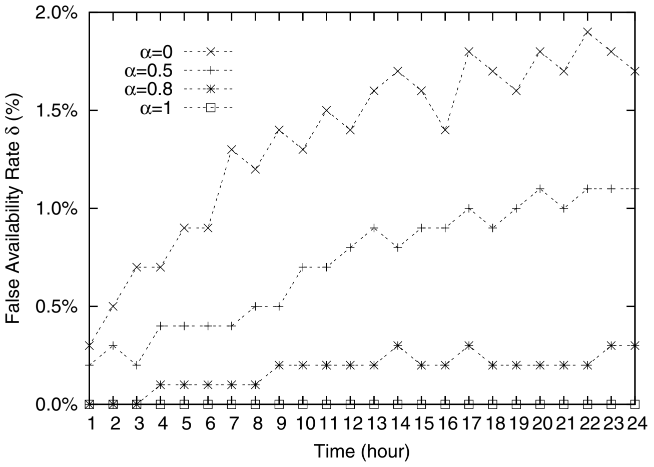


Fig. 5. Effect of varying s on the false availability rate ($\alpha = 0, 0.5, 0.8, 1$)

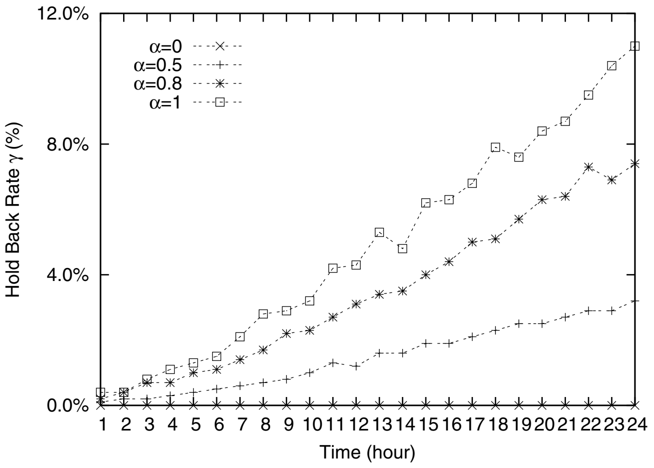


Fig. 6. Effect of varying s on the holdback rate ($\alpha = 0, 0.5, 0.8, 1$)

the holdback rate γ increases while the false availability rate δ decreases, as expected. One might be tempted to reduce timespan s in order to minimize both false availability rate and holdback rate. However, this is not a practical solution. Reducing timespan means that the e-ticket system enforces an inadequate period of time for customers to reserve tickets, who may then find that the reservations expire after a short period of time, and cannot be very happy about it. Therefore, any practical e-ticket system should not use a very small timespan s. On the other hand, if the timespan is too large (which means customers are allowed to hold tickets for a long time), the holdback rate goes up; it is more likely that more customers are unable to purchase the tickets which are in fact available. This highlights the need for making s adaptive to the demands on the tickets.

For the *no-hold* approach, *i.e.,* when $\alpha = 0$, γ becomes 0 but δ is high. For the *fully-hold* approach, *i.e.,* when $\alpha = 1$, δ becomes 0 while γ gets rather high. When $\alpha \geq 0.8$, the false availability rate δ is much lower than the *no-hold* approach while the holdback rate is also greatly reduced compared to the *fully-hold* counterpart.

Discussion. We have presented several results from our experimental study of our reservation model. First, the results verify that our approach clearly outperforms the *no-hold* and *fully-hold* approaches adopted by existing e-ticket systems. With a lower holdback rate compared with *fully-hold* model, it has neglectable false availability rate in contrast to the *no-hold* approach. Second, we find that when $\alpha \geq 0.8$, the false availability rate is almost 0 in all cases. These suggest how e-ticket systems may choose purchase rate α and adjust reservation timespan s.

4 Concluding Remarks

We have investigated the false availability and holdback problems in connection with existing online ticket booking systems. To rectify these we have proposed a transaction model that supports adaptive customer reservations with neglectable false availability and lower holdback rates. To efficiently implement the model we have developed a combination of techniques such as the analysis of purchase rate, hypothetical queries, triggers, and a method for computing reservation timespan based on demands on tickets. As verified by our preliminary experimental study, our model and methods significantly improve the existing e-ticket systems. To the best of our knowledge, this work is the first effort for reducing both the false availability and holdback rates. It yields a practical approach for providing effective support for customer reservations in e-commerce systems, including but *not limited* to e-ticket systems.

A large number of online ticket booking systems have been developed. We surveyed 30 popular e-ticket systems, including Expedia [2], Orbitz [3], Priceline [4], MakemyTrip.com, Amadeus.net as well as American Airline, Continental, Southwest, United, and US Airways [1]. As remarked in Section 1, these services only support limited customer reservations. Different virtual travel agencies offer tickets with substantially different prices and conditions for the same customer request. We found that ticket prices may vary by as much as 18% across these agencies. Thus it is desirable for customers to make reservations with a locked price beforehand.

There has been work on a variety of aspects of e-ticket systems. The need for e-ticket systems to interact with airline, hotel and payment services was advocated in [12]. An approach for building a virtual travel agency by composing "hotel booking" and "flight booking" services was proposed in [13]. A prototype for a virtual travel agency was developed in [14], based on ontology and semantic web. An enhanced user interface for B2C booking systems was proposed in [15], by means of a virtual intermediate travel agent. There has also been a host of work on generic Web services, notably on Web service compositions (*e.g.,* [10,9]).

However, to our best knowledge, no prior work has studied the false availability and holdback problems associated with existing e-ticket systems.

Airlines have to deal with reservation cancellations and no-show-ups at flight departure on a daily basis. To avoid revenue loss, most airlines compromise cancellations and no-show-ups by over-booking flights, i.e., booking excessive seats above the physical airplane capacity. There has been previous work on optimizing over-booking [16] [17] [18], which attempts to accurately estimate the number of cancellations and no-show-ups. This differs from our work in that our model provides an adaptive reservation timespan for customers to *hold* the tickets with a fixed price. Although over-booking increases seat availability, they do not allow the customers to purchase their tickets with the price they agreed upon after various reservation timespans. Furthermore, our work aims to facilitate the composition of web services for booking a travel package online. Customers can hold (reserve) a ticket while inspecting successive components or legs of the travel package. They can examine each component one by one, reserving a ticket with a fixed price before proceeding to the next component; and finally, they can "optimize" various picks, combine them to get a reasonable composition of the entire package, without worrying about the availability of tickets reserved for previous legs or the hiking-up of the price of those tickets. In contrast, to the best of our knowledge no airlines provide such a functionality. In essence, our reservation model is customer-oriented, i.e., to serve the best interest of customers, while over-booking is airline-oriented, for the best interest of airline business. The latter does not help with composition of online booking web services.

Hypothetical queries have proven useful in a wide range of applications such as decision support, version management, active databases, integrity maintenance and recently XML updates [19,5,20,21,6]. In this work we leverage hypothetical queries to reduce the overhead of transaction management incurred by customer reservations, and capitalize on the implementation technique proposed by [6] to efficiently support query evaluation (Section 2.2).

There is naturally much more to be done. First, to compute reservation timespan one might also want to take into account other information, such as promotion and sale by airlines, beyond what we have considered in Section 2.2. Second, more experiments should necessarily be conducted on real-world data, *e.g.,* real-life patterns of customer arrival and ticket booking. Third, transaction control is far more intriguing and thus deserves a full treatment for e-ticket services that are built via compositions of other services. Finally, it is interesting and practical to investigate the specific need and requirements for providing the reservation functionality for, *e.g.,* finance services.

References

1. Mitchell, C., Newton, J., Willdorf, N., Bennett, A., Stellin, S.: A to z guide to travel secrets you need to know (2007), http://www.travelandleisure.com/articles/a-to-z-guide-to-travel-secrets-you-need-to-know
2. Expedia.com, http://www.expedia.com

3. Orbitz, http://www.orbitz.com/
4. Priceline.com, http://tickets.priceline.com
5. Bonner, A.J.: Hypothetical datalog complexity and expressibility. Theoretical Computer Science 76, 3–51 (1990)
6. Griffin, T., Hull, R.: A framework for implementing hypothetical queries. In: SIGMOD (1997)
7. Sistla, A.P., Wolfson, O.: Triggers on database histories. IEEE Quarterly Bulletin on Data Engineering, Special Issue on Active Databases 15, 48–51 (1992)
8. Ramakrishnan, R., Gehrke, J.: Database Management Systems, McGraw-Hill Higher Education. McGraw-Hill, New York (2000)
9. Berardi, D., Calvanese, D., Giacomo, G.D., Hull, R., Mecella, M.: Automatic composition of transition-based semantic web services with messaging. In: VLDB, Trondheim, Norway (2005)
10. Berardi, D., Giacomo, G.D., Lenzerini, M., Mecella, M., Calvanese, D.: Synthesis of underspecified composite e-services based on automated reasoning. In: ICSOC (2004)
11. Willig, A.: A short introduction to queueing theory, http://www.tkn.tu-berlin.de/curricula/ws0203/ue-kn/qt.pdf
12. Haas, H.: Web service use case: Travel reservation. W3C (2002)
13. Pistore, M., Roberti, P., Traverso, P.: Process-level composition of executable web services: "on-the-fly" versus "once-for-all" composition. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, Springer, Heidelberg (2005)
14. Zaremba, M., Moran, M., Haselwanter, T.: Applying semantic web services to virtual travel agency case study. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, Springer, Heidelberg (2006)
15. Malizia, A.: Adding flexibility to b2c booking systems using a virtual intermediate travel agent. In: HVL/HCC (2005)
16. Leder, K.Z., Spagniole, S.E., Wild, S.M.: Probabilistically optimized airline overbooking strategies, or "anyone willing to take a later flight?!". The UMAP Journal (2002)
17. Klophaus, R., Pölt, S.: Airline overbooking with dynamic spoilage costs. Journal of Revenue and Pricing Management (2007)
18. Lawrence, R.D., Hong, S.J., Cherrier, J.: Passenger-based predictive modeling of airline no-show rates. In: KDD 2003. Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press, New York (2003)
19. Balmin, A., Papadimitriou, T., Papakonstantinou, Y.: Hypothetical queries in an olap environment. In: VLDB (2000)
20. Fan, W., Cong, G., Bohannon, P.: Querying XML with update syntax. In: SIGMOD (2007)
21. Gabbay, D.M., Giordano, L., Martelli, A., Olivetti, N.: A language for handling hypothetical updates and inconsistency. Journal of IGPL 4, 385–416 (1996)

A Multi-level Model for Activity Commitments in E-contracts

K. Vidyasankar[1,*], P. Radha Krishna[2], and Kamalakar Karlapalem[3]

[1] Department of Computer Science, Memorial University, St. John's, Canada, A1B 3X5
vidya@cs.mun.ca

[2] Institute for Development and Research in Banking Technology, Hyderabad, India
prkrishna@idrbt.ac.in

[3] International Institute of Information Technology, Hyderabad, India
kamal@iiit.ac.in

Abstract. An *e-contract* is a contract modeled, specified, executed, controlled and monitored by a software system. A *contract* is a legal agreement involving parties, activities, clauses and payments. The goals of an e-contract include precise specification of the activities of the contract, mapping them into deployable workflows, and providing transactional support in their execution. Activities in a contract are complex and interdependent. They may be executed by different parties autonomously and in a loosely coupled fashion. They may be compensated and/or re-executed at different times relative to the execution of other activities. Both the initial specification of the activities and the later verification of their executions with respect to compliance to the clauses are tedious and complicated. We believe that an e-contract should reflect both the specification and the execution aspects of the activities at the same time, where the former is about the composition logic and the later about the transactional properties. Towards facilitating this, we propose a multi-level composition model for activities in e-contracts. Our model allows for the specification of a number of transactional properties, like atomicity and commitment, for activities at all levels of the composition. In addition to their novelty, the transactional properties help to coordinate payments and eventual closure of the contract.

1 Introduction

An *electronic contract*, or *e-contract* in short, is a contract modeled, specified, executed, controlled and monitored by a software system. A *contract* is a legal agreement involving parties, activities, clauses and payments. The activities are to be executed by the parties satisfying the clauses, with the associated terms of payment.

Consider, for example, a contract for building a house. The parties of this contract include a customer, a builder, a bank and an insurance company. The contract has several parts: (a) The builder will construct the house according to the specifications

* This research is supported in part by the Natural Sciences and Engineering Research Council of Canada Discovery Grant 3182.

of the customer. Some of the activities such as carpentry, plumbing and electrical work may be sub-contracted; (b) The customer will get a loan for the construction from the bank. He will apply for a mortgage, and work out details of payment to the builder, directly by the bank, after inspection of the work at multiple intervals; and (c) The house shall be insured comprehensively for the market value covering fire, flood, etc. in the joint names of the bank and the customer. The activities of the customer and the builder include the following.

- Customer: (i) submitting the loan application, (ii) setting up coordination between bank and builder, (iii) receiving payments, and (iv) arranging monthly repayments.
- Builder: (i) scheduling different works involved in the construction and procuring raw material, (ii) building the house as per the agreement, (iii) giving part of the work to sub-contracts, if any, (iv) receiving the payments, (v) making payments to its staff and sub-contract parties, if any, and (vi) handing over the constructed house to the customer.

An example of a clause relating to payments can be, in verbatim, as follows.

"If the bank is of the opinion that the progress of work of construction of the said house is unsatisfactory, the bank shall be at liberty to decline to make payment of any undisbursed installment of the said loan or at its discretion postpone the payment thereof until such time the bank is satisfied that the cause or causes for its dissatisfaction with the progress and quality of work has or have been removed and the bank shall incur no responsibility or liability to the borrower either in damage or otherwise for declining to make payment or postponement of payment of any undisbursed installment as aforesaid."

Majority of contracts in real world are documents that need to be gleaned to come up with e-contract specifications that are executed electronically. The execution can also be fairly complex. The goals of the e-contract include precise specification of the activities, mapping them into deployable workflows, and providing transactional support in their execution.

1.1 E-contract Commitment

As seen above, contracts are complex in nature. Both the initial specification of the requirements and the later verification of the execution with respect to compliance to the clauses are very tedious and complicated. This is due, partly, to the complexity of the activities. Typically, the (composite) activities are interdependent with other activities and clauses. They may be executed by different parties autonomously, in a loosely coupled fashion. They are long-lasting. Though the desirable outcomes of their executions are stipulated in the contract specification, their executions may yield unexpected results. This might result in re-design and even re-specification of the contract.

Another major reason for the increased complexity is the variety in the "execution states" of the activities. In database applications, *atomicity* is strived for in a (simple) transaction execution. That is, a transaction is executed either completely or (effectively) not at all. Partial execution is rolled back. On successful completion, the transaction is *committed*. In multi-database and other advanced database applications, transactions may be committed (locally) and then rolled back logically, by executing compensating transactions. This property is called *compensatability.* The property of repeatedly executing a transaction until successful completion is also considered; this is called *retriability.* In e-contract activities also, both compensatability and retriability properties are encountered for the activities, and in fact, in more sophisticated ways. For example,

(i) Both complete and partial executions may be compensated,
(ii) Both successful and unsuccessful executions may be compensated,
(iii) Even "committed" executions may be retried,
(iv) Retrying may mean, in addition to re-execution, "adjusting" the previous execution, and
(v) Activities may be compensated and/or re-tried at different times, relative to the execution of other activities.

Example 1: An instance of (iii) is the following. A pipe is fixed correctly as specified in the contract. A month later, the pipe breaks while constructing a mini-wall in the balcony. As per the clause stated in the contract 'any damage or loss of goods/material during construction of house is the responsibility of the builder and the builder has to repair or replace at no additional cost', the builder has to fix the pipe. An instance of (iv) is, in the process of re-payment of a bank loan, if a check is bounced for some reasons, the customer has to pay penalty in addition to the actual amount.

We assert that a key to handle the complexity of a contract execution is adherence to transactional properties. In this paper, we subscribe to the notion that the activities should be specified in an e-contract such that their execution embodies transactional properties. We start with basic activities and construct composite activities hierarchically. In the first level, a composite activity consists of basic activities; in the next level, a composite activity consists of basic and/or composite activities of level one; etc. The highest level activity will correspond to the "single" activity for which the contract is made. We call this the *contract-activity.* (We note that there could be multiple contracts for a single activity. For example, for building a house, there could be separate contracts between (i) customer and the builder, (ii) customer and the bank, (iii) customer, bank and insurance company, etc. These contracts will be related. We consider this set of contracts as a part of a single high level contract whose contract-activity is building the house.) Then, our contention is that the execution of each activity, at every level, should satisfy transactional properties. Towards facilitating this, we propose a *multi-level composition model* for activities in e-contracts.

The two main properties that are relevant for our work are atomicity and commitment. For atomicity, either a complete successful execution or an (effectively) null execution should be obtained. Given a non-null, partial execution, the former is obtained by *forward-recovery* and the latter by *backward-recovery.* The retriability and compensatability properties relate to whether forward-recovery or backward-recovery can be carried out. For activity at each level, we consider successful execution, atomicity, compensatability, retriability, backward-recovery and forward-recovery properties. We then define commitment of the activities based on these properties. We do this uniformly, the same way irrespective of the level of the activity.

Every activity in the contract must be *closed* at some time. On closure, no execution related to that activity would take place. The closure could take place on a complete or incomplete execution, and on a successful or failed execution. On closure of the *contract-activity*, the e-contract itself can be *closed.* The e-contract closure is mostly a human decision. It involves settlement, of payment and other issues, between the parties. Our composition model helps to streamline closure of the e-contract also.

E-contract closure is also referred to as *e-contract commitment*. We use the term *e-contract commitment logic* to refer to the entire logic behind the commitment of the various activities of the e-contract, and the closure of the activities and the e-contract.

1.2 Related Work

Considerable work has been carried out on the representation of e-contracts and developing e-contract architectures. We refer to some of them in the following. E-ADOME [5] and CrossFlow [6] systems describe the workflow interfaces as activities and transitions in e-contracts. In the same way, Chiu et al. [1] develop a framework for workflow view based e-contracts for e-services. A rule-based approach is presented in [4] to deal with exceptions raised during e-contract execution. Green and Vonk [3] describe the relationship between transaction management systems and workflows for transactional business process support.

Xu [14] proposes a pro-active e-contract monitoring system to monitor contract violations. They represent constraints using propositional temporal logic in order to provide formal semantics for contract computation at the contract fulfillment stage. However, their formalism does not provide the execution level semantics of an e-contract commitment. Business Transaction Framework based on Abstract Transactional Constructs, developed by Wang et al [13], provides a specification language for identifying and interpreting clauses in e-contracts. To the best of our knowledge, adequate formalism for e-contract commitment does not exist in the literature.

1.3 Contributions and Organization of the Paper

In this paper, we propose a framework for e-contract commitment. We do this by developing a multi-level specification model, also called composition model, of the (composite) activities of a contract, and defining transactional properties for the activities at every level. Transactional properties have been defined to suit the real world, non-electronic, activities. The salient points are the following.

i. Transactional properties are defined for executions of activities rather than activities themselves. This accounts for the fact that different executions of the same activity might have different characteristics.
ii. Atomicity is defined for executions of composite activities of any level in spite of the executions of even some basic activities being non-atomic. This helps in dealing with backward- and forward-recoveries at each level independent of its descendent levels.
iii. The scope of retriability is extended from executing the same activity again, or executing some other substitute activity, to adjustments to the original execution.
iv. Two levels of commitment, weak and strong, are defined. On weak commitment, the execution becomes non-compensatable, and on strong commitment it

becomes non-retriable. Weak commitment is the commitment property of the traditional database operations and the pivotal property of multi-database operations. The strong commitment property definition is new.

Both (a) defining transactional properties for activities of a contract and (b) influencing e-contract design with transactional properties are novel and have not been done before.

The rest of the paper is organized as follows. Section 2 describes the e-contract commitment aspects. We present the basic concepts related to our model in Section 3 and the model in Section 4. Section 5 presents the discussion and concludes the paper.

2 E-contract Commitment

Transactional semantics, workflow semantics, clauses and payment components of e-contract need to be considered for addressing e-contract commitments. Workflow semantics deals with the composition logic, namely, the semantics of the executions of the individual activities that constitute the workflow. Transactional semantics deals with the commitment logic, about atomicity, forward- and backward-recovery and commitment of the executions, and closure of the activities and the e-contract. Both clauses and payments influence, and are influenced by, both the workflow and transaction semantics.

2.1 ER^{EC} Architecture

An ER^{EC} framework has been developed in [7, 8] for modeling and enactment of an e-contract. It is shown in Figure 1. The ER^{EC} meta-schema constitutes the meta-layer whereas the ER^{EC} data model, Activity Party Clauses (APC) and Activity Commit Diagrams (ACDs) constitute the conceptual layer. The ER^{EC} data model and APC constructs help in specifying the workflows for activities of an e-contract. The ACDs facilitate monitoring workflow execution based on the specifications provided in the contract, as well as the exceptions that may occur during the execution of the workflows. All the components at the conceptual layer form the basis for arriving at commitment specifications. They provide activity-commitment semantics based on the contract document. Further execution level semantics are provided by workflow instances. The commitment specifications facilitate specifying the semantics for transactional support, activity commitment and workflow commitment, at the logical level.

Commitment specifications were not addressed in our model described in [8] for e-contracts. In that paper, Figure 1 was given without the commitment specifications box.

2.2 Our Approach

Compensatability and retriability properties were first identified in the context of atomicity of multi-database applications (for instance, [10]). To achieve atomicity (of a global transaction) in autonomous execution (of the subtransactions), a multi-database transaction is modeled to consist of a sequence of compensatable

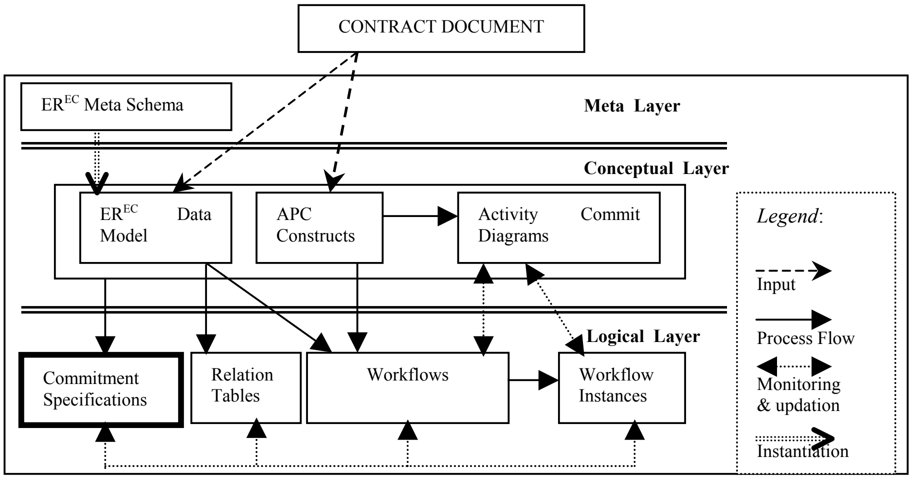


Fig. 1. ER^{EC} architecture for specification and execution of e-contract

transactions, followed possibly by a *pivotal* (that is, non-compensatable) transaction and a sequence of retriable transactions. In particular, each multi-database transaction can have at most one pivot. Schuldt et al. [9] extended this idea to transactional processes by allowing multiple pivots. Clearly, with multiple pivots, atomic execution may not be possible (when some pivots are executed but others cannot be executed). They defined a property, called *guaranteed termination,* which formalized "graceful" termination of the transaction after some pivots were executed. In addition, the pivots in a guaranteed termination were executed in sequence. Further extension was done in [11, 12], in the context of composition of Web Services. First, (i) the guaranteed termination concept was extended to atomicity (of global transaction, or composite activity or service), (ii) forward- and backward-recovery procedures for achieving atomicity were given, and (iii) non-sequential, tree-like, execution of the pivots was accommodated. Then the transactional properties (atomicity, compensatability, retriability and pivot) were extended to hierarchically composed activities/services. It was shown that the transactional properties can be considered at each level independently of the properties of the other level activities.

The proposal in this paper is along the lines of [11, 12] but tailored to e-contract environment.

The model in this paper will form a part of the commitment specifications.

3 Basic Concepts

In this section, we present the concepts and notations relevant for transactional properties in the context of e-contracts, and in the next section we present our model.

3.1 Basic Activities

We consider certain activities as *basic* in our model. Typically, these are the activities which cannot be decomposed into smaller activities, or those that we want to consider in entirety, and not in terms of its constituent activities.

In e-contract environment, whereas some basic activities may be executed 'electronically' (for example, processing a payment), most others will be non-electronic (for example, painting a door). We desire that all basic activities are executed atomically, that is, it is either not executed at all or executed *completely*. However, incomplete executions are unavoidable and we consider them in our model.

3.2 Constraints

Each activity is executed under some constraints with respect to who can execute, when it can be executed, which executions are acceptable, etc. The acceptability may depend on whether the activity can be executed within a specified time period, cost of execution, compensatability or other transactional properties, side-effects, etc.

A complete execution of an activity that satisfies all the constraints specified for the execution of that activity *at the time of its execution* is called a *successful termination*, abbreviated *s-termination*, of that activity. The constraints themselves are specified in terms of an *s-termination predicate*, or simply, *st-predicate*. A complete execution which does not satisfy the st-predicate is called a *failed termination*, abbreviated *f-termination*. The s- and f-termination distinction is applied to incomplete executions also, depending on whether the st-predicate is satisfied thus far.

Example 2: Consider the activity of painting a wall. The execution is incomplete while the wall is being painted, and complete once the painting is finished. If the paint job is good at the end (respectively, in the middle), the execution is a complete (respectively, incomplete) s-termination. If the paint job is not satisfactory, we get a complete or incomplete f-termination. The st-predicate specifying the goodness of the job could be: (i) one undercoat and one other coat of paint and (ii) no smudges in the ceiling or adjacent walls.

The constraints may change, that is, the st-predicate of an activity may change, as the execution of the contract proceeds. (In the example above, two coats of paint may be required in addition to undercoat.) Such changes may invalidate a previous complete execution of that activity. When this happens, the execution needs to be adjusted.

3.3 Compensatibility

One of the ways an execution can be adjusted is by compensation, that is, nullifying the effects of the execution. We look at compensation as a logical roll back of the original execution. Absolute compensation may not be possible in several situations. In some cases, the effects of the original execution may be ignored or penalized and the execution itself considered as compensated. Compensation may also involve execution of some other, *compensating*, activity. Inability to execute a compensating activity within a prescribed time limit may also make the original execution non-compensatable.

It is possible that an execution can be compensated within a certain time, but not afterwards. The time could be "real" time (for example, flight reservations can be cancelled without penalty within 24 hours of booking, and vinyl flooring glued to the floor can be removed before the glue sets) or specified relative to the execution of some subsequent activities (for example, flight bookings can be cancelled until paid for, and a (stolen) cheque can be cancelled before it is cashed).

Note that we do not attribute compensatability property to an activity, but only to an execution of that activity. For the same activity, some executions may be compensatable, whereas others may not be. For example, when we book flight tickets we may find that some tickets are non-refundable, some are fully refundable, and some others partially refundable. Purchasing a fully refundable ticket may be considered to be a compensatable execution, whereas purchasing any other type of ticket could be non-compensatable. Thus, compensatability of the execution (purchasing a flight ticket) may be known only during execution, and not at the specification time of the activity.

3.4 Retriability

Another way of adjusting an execution is by *retrying*. By retriability, we mean the ability to get a complete execution satisfying the (possibly new) st-predicate. It is possible that the original execution is compensated fully and new execution carried out, or the original execution is complemented, perhaps after a partial compensation, with some additional execution, for instance, the second coat of painting in Example 2.

Retriability may also be time-dependent. It may also depend on the properties of execution of other preceding, succeeding or parallel activities. Again, in general, some executions of an activity may be retriable, and some others may not be retriable.

We note that retriability property is orthogonal to compensatability. That is, an execution may or may not be retriable, and, independently, may or may not be compensatable.

3.5 Execution States

We assume that the state of a complete s-terminated execution changes in the following sequence:

(a) It is both compensatable and retriable;
(b) It becomes non-compensatable, but is still retriable; and
(c) It becomes (non-compensatable and) non-retriable.

We note that in state (a) the execution may be compensated and/or retried several times. Similarly, in state (b), the execution may be retried several times, before state (c) is reached. It is also possible that an (un-compensated) execution remains in state (a) and never goes to state (b), or it goes to state (b), but not to state (c).

We say that an execution is *weakly committed* if it is at state (b), that is, when it is or has become non-compensatable, and is *strongly committed* if it is at state (c). We note that both weak and strong commitments can be forced upon externally also. That is, the execution can be *deemed* as (weakly or strongly) committed, for reasons outside of that execution. An example is payment to a sub-contractor for execution of an activity, and the non-obligation and unwillingness of the sub-contractor to compensate (in case of weak commitment) or retry (in case of strong commitment) the execution after receiving the payment. We say also that an activity is *weakly* (*strongly*) *committed* when an execution of that activity is weakly (strongly) committed.

We allow compensatability and retriability properties to be applicable to incomplete executions also. We assume the first two of the above state transition sequences for partial executions. That is, a partial execution is both compensatable and retriable in the beginning, and may become non-compensatable at some stage. Then, if it is retriable, that is, a complete s-termination is guaranteed, then the execution can be weakly committed. Note that we are simply allowing the transition from uncommitted to weakly committed state to occur even before the execution of the activity is complete. We do not allow transition from weakly committed to strongly committed state until (or some time after) the execution is completed.

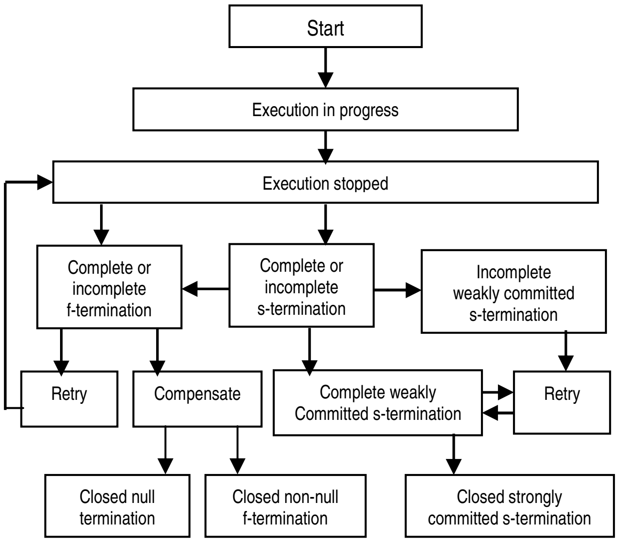

Start
Execution in progress
Execution stopped
Complete or incomplete f-termination
Complete or incomplete s-termination
Incomplete weakly committed s-termination
Retry
Compensate
Complete weakly Committed s-termination
Retry
Closed null termination
Closed non-null f-termination
Closed strongly committed s-termination

Fig. 2. Execution stages of an activity

Figure 2 depicts the execution stages (boxes) of an activity, and possible transitions (arrows) between them. Some notable points are the following.

- Retry denotes re-execution possibly after partial or full backward-recovery.
- A full backward-recovery yields the null termination. If re-execution of the activity is intended after the null termination, we take the backward-recovery as part of retry; otherwise, it is taken as compensation.
- A complete s-termination may become an f-termination, with a change in st-predicate. If this happens before weak commitment, the transitions of an f-termination are followed. Otherwise, if the execution is already weakly committed, then a retry that guarantees s-termination is assured.
- If the compensation succeeds we get the null termination. Otherwise, we get a non-null f-termination.

The “final” state of execution of a basic activity is closure. The diagram shows three possible states of closure: (i) null; (ii) non-null (incomplete or complete) f-termination; and (iii) (complete) s-termination, which also corresponds to strong commitment of the execution.

Complete and incomplete, and s- and f-terminations can be defined for composite activities also, analogously. This is done in the model. We illustrate the different categories with the following example.

Example 3: Let *U* be a composite activity consisting of (i) writing and printing a letter, (ii) preparing an envelope, and (iii) inserting the letter in the envelope and sealing it. Call the activity (ii) as *C*. Then *C* is composed of (a_1) printing the From and To addresses on the envelope, perhaps with a printer and (a_2) affixing a stamp on the envelope. Consider an execution of *U*. The following possibilities arise.

- (i) is done but (ii) fails possibly because of printing a wrong address. Now we may decide not to bother preparing a new envelope and sending the letter. This is an incomplete f-termination.
- (i) and (ii) are done. (iii) is not done (yet). This is an incomplete s-termination.
- All the three activities are done, but we realize afterwards that the address is wrong, that is, (ii) is not executed correctly. This is a complete f-termination.
- All activities have been done correctly. This is a complete s-termination.

Figure 2 is applicable to composite activities also. We explain this later.

4 Composition Model for Activities

In this section, we describe our multi-level composition model for the activities in an e-contract. We start with a specification of *one* level, the "bottom" level.

4.1 Path Model

We start with a simple model, called the *path model*, to illustrate the various key aspects. We will extend it to a general model later. Our description is in three parts – composition, execution and transactional properties. We use bold font to denote compositions, and italics to denote their executions, that is, the composite activities.

A. Composition

- Composition **C** is a rooted tree. It is a part of a higher level composition **U**.
- An st-predicate is associated with **C**. This will prescribe the s-terminations of **C**. (We define s-terminations of a composition later.)
- Nodes in the tree correspond to basic activities. They are denoted as $\mathbf{a_1}$, $\mathbf{a_2}$, etc.
- With each node in the tree, an st-predicate and a *children execution predicate*, abbreviated *ce-predicate*, are associated.
- The st-predicate specifies s-terminations of that activity. The ce-predicate specifies, for *each* s-termination of that node, a set of children from which *exactly one* child has to be executed next, the child being chosen according to a given partial order of preferences. The ce-predicate for the leaf nodes of the composition is null.
- We assume that the st-predicate and ce-predicate of each of the nodes in **C** are derived from the st-predicate of **C**.

Example 4: Figure 3 shows a composition where C_i's are construction activities for a product and I_j's are Inspection activities. After the first two stages, C_0 and C_1, of the construction, the inspection I_1 is carried out. Depending on the result, say quality of the product after C_1, C_2 is carried out if possible, and C_2' or C_2'' otherwise, in that order. This will be the ce-predicate at I_1. Only the inspection I_2 after C_2 is shown. The st-predicate for each C_i will be the guidelines to be followed for that construction. The st-predicate for each I_i will be the acceptable results of the things to be checked in that inspection.

B. Execution

- An execution of activity $\mathbf{a_i}$ is denoted a_i.
- An execution E of **C** yields a composite activity C. The execution consists of execution of activities in a path from the root to a leaf. If all the activities in this path have been executed completely, then E is a *complete* execution of **C**. Otherwise, that is, if only the activities from the root to some non-leaf node have been executed and/or the executions of some activities are not complete, then it is an *incomplete* execution of **C**. If E is a complete (incomplete) execution and each activity in E has s-terminated, then E is a *complete (incomplete) s-termination* of **C**. A complete s-termination is usually called simply as an *s-termination* of **C**. An *f-termination* of **C** is either a complete or incomplete execution in which executions of some activities have f-terminated.
- In each s-termination C, at each non-leaf node a_i, the selection of the child of a_i satisfies the ce-predicate currently specified for $\mathbf{a_i}$ in **C**.
- Both the st-predicate and the ce-predicate at each node $\mathbf{a_i}$ may be changing as the execution of subsequent activities of **C** proceeds.
- Partial execution of **C** will be represented by a path from the root a_1 to some node a_i in the tree, and will be denoted (a_1, ..., a_i), and also as $C_{[1,i]}$. Here, the part that is yet to be executed to get a complete termination of **C** is the subcomposition of **C** from $\mathbf{a_i}$, called the *suffix* of **C** from $\mathbf{a_i}$, denoted $\mathbf{C_{[i]}}$. The subcomposition will contain the subtree of **C** rooted at $\mathbf{a_i}$, with the st-predicate and ce-predicate of $\mathbf{a_i}$ adjusted according to the execution $C_{[1,i]}$, and the st-predicate and ce-predicate of all other nodes in the subtree being the same as in **C**.

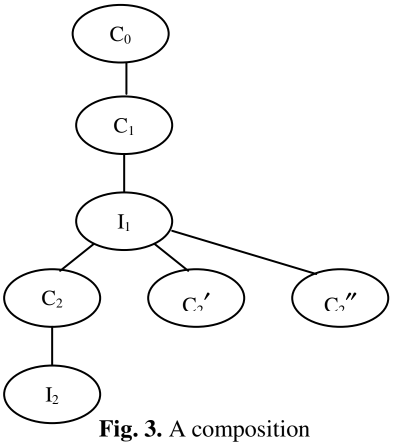


Fig. 3. A composition

- Each activity a_i in C may first be weakly committed, and then strongly committed *relative to* **C**, some time after its s-termination.
- Once a_i is weakly committed, as stated earlier, it cannot be compensated, and once it is strongly committed, it cannot be retried. Again, both compensatability and retriability are relative to **C**. We elaborate this later.
- The activities in C are (both weakly and strongly) committed in sequence. That is, when a_i is weakly committed, *all* activities that precede a_i in C and have not yet

been weakly committed are also weakly committed. Similarly, strong commitments of the executions are also in sequence.

C. Transactional Properties

- Composition **C** assumes that each of its activities a_i is executed atomically. Thus an incomplete f-termination of a_i is assumed to be compensatable, to get an effective null execution, relative to **C.**
- The execution of the entire composition **C** is intended to be atomic in **U**. Therefore, an execution of **C** should eventually yield a complete s-termination or the null termination.
- Consider an execution E of **C**.
  - If E is an incomplete s-termination, then forward-recovery is carried out by executing the suffix of E in **C** or a different acceptable sub-composition, to get a complete s-termination.
  - If E is either incomplete or complete f-termination, then the executions of some activities may have to be adjusted (partial backward-recovery) to get an incomplete s-termination, and a forward-recovery is carried out.
  - To get the null termination, E has to be compensated. This is the full backward-recovery.

D. Implementation Issues

(a) Partial Backward-Recovery
It starts with re-executing a_j, for some $j \leq i$, where a_i is the latest activity that has been or being executed. If a_j has to be compensated, then all activities in the execution following a_j are also compensated, and a different child of a_{j-1} is chosen with possibly an updated ce-predicate at a_{j-1}. If a_j is retried, then, after retrying, a_{j+1} may need to be compensated or retried. Continuing this way, we will find that for some k, $k \geq j$, the activities in the sequence (a_j, …, a_{k-1}) are retried and those in (a_k, …, a_i) are compensated. This is illustrated in Figure 4.

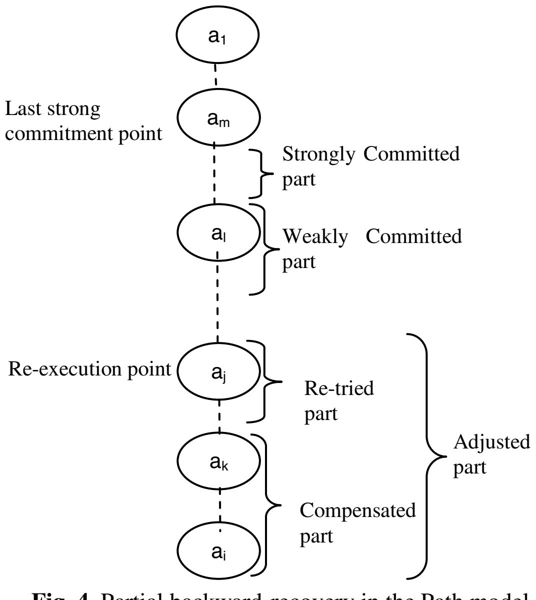


Fig. 4. Partial backward-recovery in the Path model

We note that if m is the largest index such that a_m is strongly committed, then $j > m$, and if n is the largest index such that a_n is weakly committed, then $k > n$. This follows since, by the definitions of strong and weak commitments, executions of activities up to a_m cannot be retried and those up to a_n cannot be compensated. In the figure, a_n is not shown. It will be between a_m and a_k.

Similar to the abort, commit and other dependencies in [2], we can define dependencies between activities a_p and a_q:

- If a_p is compensated then a_q must be compensated/retried; and
- If a_p is retried then a_q must be retried.

Then the re-execution point will be the earliest point in the execution based on the transitive closure of these dependencies. We also note that dependencies of the type

- If a_p is compensated/retried then a_q must be weakly/strongly committed

might also exist. This would require some activities to be strongly committed and/or some activities to be weakly committed (on re-execution of a_j). This is also shown in Figure 4.

The following example illustrates backward-recovery.

Example 5: In the composition of Figure 3, suppose C_2 was executed after I_1, and I_2 fails. It may be decided that the product be sent back to C_1 for some adjustment and inspected, and the options C_2' and C_2'' explored. This would amount to rolling back I_2 and C_2, and re-executing C_1 and I_1, each with adjusted st-predicate. Here the adjusted ce-predicate for I_1' will have only C_2' and C_2'' options. Also, retrying C_1' might require strong commitment of C_0.

(b) Point of Commitment
The execution of an activity a_i can be weakly committed any time, and then, after an s-termination, can be strongly committed any time. Weak commitment immediately after the s-termination gives pivotal property in the traditional sense. Waiting until the end of the execution of the entire composite activity will give the compensatability and retriability options until the very end. The longer the commitment is delayed, the more flexibility we have for adjustment on execution of the subsequent activities. However, commitment of some subsequent activities may force the commitment of a_i.

(c) Adaptivity
As mentioned earlier, the ce-predicate will keep changing as the execution proceeds. Also, additional execution paths can be added, as descendents of a node, in the middle of the execution of the composite activity. Some execution paths may be deleted too. Thus, the composition could be adaptive and dynamic.

4.2 Tree Model

We now present an extension, called the *tree model.* Here, we consider compositions that allow for more than one child to be executed at non-leaf nodes. Therefore, the execution yields a tree, instead of just a path, as a composite activity. The features of this model are essentially the same as in the path model. The difference is only in the complexity of the details. We outline the details in the following.

A. Composition

Here also, a composition **C** is a tree and it is a part of a higher level composition **U**. An st-predicate is associated with **C**. An st-predicate and a ce-predicate are associated with each node. These will be derived from the st-predicate of **C**. The ce-predicate is null for all leaves of **C**. The ce-predicate at non-leaf nodes may be sophisticated.

- More than one child may be required to be executed.
- In general, several sets of children may be specified with the requirement that one of those sets be executed.
- These sets may be prioritized in an arbitrary way.
- Execution of children within a set may also be prioritized in an arbitrary way.

B. Execution

A composite activity *C* is a subtree of **C** such that

- it includes the root, some leaves of **C**, and all nodes and edges in the paths from the root to those leaves in **C**, and
- the execution of each node satisfies the st-predicate prescribed for that node, and the children of each non-leaf node of the subtree satisfy the ce-predicate specified in **C** for that node.

A partial execution *E* of **C** will be represented by a subtree of **C**, called *execution-tree*, consisting of all the nodes of **C** that have been executed so far and the edges between them. The suffix of the execution *E* can be defined similar to that in the path model. It will consist of sub-trees of **C** rooted at some of the leaves of the execution tree, with st- and ce-predicates properly adjusted.

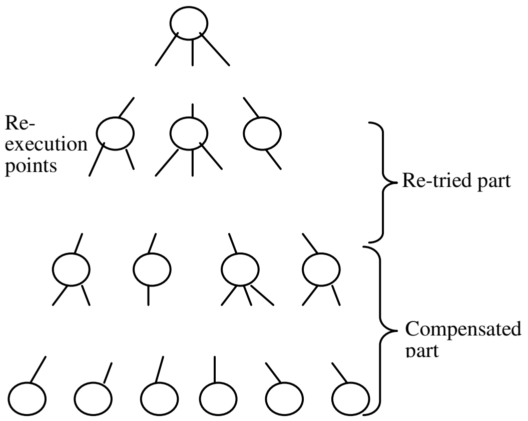


Fig. 5. Partial backward-recovery in the Tree model

C. Transactional Properties

Again, the execution of the entire composition **C** is intended to be atomic in **U**. Forward-recovery of *E* will consist of execution of the suffix of *E*. Partial backward-recovery of *E* will again consist of retrying the executions of some of the activities of the execution-tree, and compensating some others. This is illustrated in Figure 5.

4.3 Multi-level Model

So far, we have dealt with compositions at a single level, in fact, the bottom-most level where all activities are basic activities. Now we extend the model by allowing basic or composite activities in the compositions. This gives us a multi-level, hierarchical, composition model. The highest level activity is the contract-activity. In the previous sections, a composition **C** is described as a tree. An execution of **C** yields a composite activity *C*, which is a path graph in the path model and a tree in the tree model. We call (both of) them a *composite activity tree,* or *c-tree* in short.

An outline of the multi-level model is the following.

A. Composition
A composition **C** is a tree as in the tree model. Nodes in the tree are (sub)compositions of basic or composite activities. Compositions of composite activities are, again, trees as in the tree model. Thus **C** is a "nested" tree. An st-predicate is associated with **C**.

B. Execution
Execution of each subcomposition of **C** yields a c-tree. (For a basic activity, the c-tree will have just one node.) To put these trees together, we use the following notation. A c-tree is converted to a one source one sink acyclic graph by adding edges from the leaves of the tree to a single (dummy) sink node. We call this a *closed c-tree.*

In the execution of a multi-level composition **C**, at the top level we get a closed c-tree with nodes corresponding to the executions of activities in **C**. Each of the activities will again yield a closed c-tree. Thus, the graph can be expanded until all the nodes correspond to basic activities.

Partial execution is considered as in the tree model, level by level, in nested fashion.

C. Transactional Properties
At each individual level, for each node, the transactional properties discussed with the tree model are applicable. After the recovery of one node, the recovery efforts at the parent level execution will continue.

We have already discussed s-terminations and f-terminations of composite activities. We now consider compensatability and retriability of composite activities. Compensation, in general, amounts to execution of a compensating activity, that is, execution of a composition that corresponds to compensation. (For example, if the original execution is building a garden shed in the backyard, the compensation might be the demolition of that shed.) This compensation will also be specified as a tree with suitable st-predicate. Retrying a composite activity involves, in the general case, a possible backward-recovery followed by forward-recovery. The forward-recovery part can be accommodated by adding additional subtrees at some nodes and specifying the st- and ce-predicates for the nodes in them, and adjusting the ce-predicates of other nodes appropriately.

Thus, in general, re-execution of a composite activity would require adjusting the composition of that activity in terms of adding and/or deleting some nodes and adjusting the st- and ce-predicate of the nodes. This can also be thought of as coming up with a new composition for that activity, mapping the previous execution on the new composition, identifying the s-terminated part, and doing a backward- and/or forward-recovery. The re-execution and adjustments of the st- and ce-predicates will then be top-down.

4.4 Multi-level Commitment and Closure

(a) Commitment
As we referred to earlier, suppose **C** is a composition corresponding to a composite activity of **U**, and $\mathbf{a_i}$ is the composition of an activity of **C.** Let a_i, C and U be the

respective executions. We have defined the compensatability and properties of a_i to be relative to **C**. Similarly, compensatability and retriability of *C* will be relative to ***U*.**

Example 6: In Example 3, suppose the addresses are printed and the stamp glued, and we find later that the To address is incorrect. If the affixed stamp cannot be removed, the activity a_2 is non-compensatable, but only relative to *C*. The activity *C* itself may be compensatable relative to *U*, amounting to just tearing up the envelope and bearing the loss of the stamp. Then, though a_2 itself is not compensated the composite activity containing a_2 is compensated.

Similarly, the commitment properties at the two levels are also independent of each other. We give two examples. (1) Activity a_i could be strongly committed, meaning that it cannot be compensated or re-executed in *C*, but *C* itself may be weakly committed relative to *U*, meaning that it may be re-executed perhaps with additional activities. *C* could be weakly committed even if some activities of *C* are not executed yet, if retrying of *C* can be carried out by compensating the current execution completely and re-executing it to get an s-termination. (2) An example of a_i being weakly committed and *C* being strongly committed is that of fixing (perhaps in the warranty period) a broken pipe after the construction of the house is finished and the builder paid fully. Thus our model allows, as mentioned in Section 1, re-executing even a “committed” activity, by dealing with commitment in multiple levels.

(b) Closure of Composite Activities

A composite activity *C* also can be closed in three different states depicted in Figure 2, namely, null termination, (incomplete or complete) non-null f-termination, and (complete) strongly committed s-termination. The null execution might be the result of executing a compensating activity. Therefore, in any of these terminations of *C,* the constituent activities of *C* might be closed in any of the three terminations. Now, *C* may be closed either before or after some or all of the constituent activities of *C* are closed. An example of the former would be not waiting for the closure, or even termination, of some activities that compensate some other activities in the original execution of *C,* that are guaranteed to succeed.

(c) Closure of E-contract

Some of the activities (usually high level ones) will correspond to parts of the contract or subcontracts. As noted earlier, at the highest level, the composition is for the entire contract-activity. On closure of such activities, the corresponding contracts themselves might be closed. Closure of a contract intuitively refers to expiring the “life” or validity of the contract. For example, a contract for building a house may close after the warranty period during which the builder is responsible for repairs. A sub-contract for maintaining an air-conditioning system installed in that house may close at a different time. The transactional properties in our model can be used to refine the conditions for closure of the contracts.

4.5 Implementation Issues

All the issues discussed in the path model section are applicable here also. We discuss some additional issues in the following.

We have associated an st-predicate and a ce-predicate with each activity in our model. They are activity-dependent. We can expect that they can be expressed more precisely for some activities than for some others. In fact, for some activities, what constitutes s-termination may not be known until after an execution of that activity, and even after the execution of many subsequent activities. We note also that the st-predicate of a composite activity determines the st-predicate and the ce-predicate of its constituent activities. Hence, specification of the st- and ce-predicates is crucial. This will be the role of the (activity and) workflow semantics.

Whereas the semantic specification of ce-predicate would be application-dependent, syntactic specification may be made more precise, with an appropriate language. We can expect that such a language would have constructs for specifying priorities and Boolean connectives. An example is booking an all (flight-hotel-food) inclusive package, and if it is not available then booking flights and three-star hotels separately, for a vacation.

The ce-predicate allows specifying preferences in the selection of the children activities to be executed. Preferences may exist for s-terminations too. This may depend on functional as well as non-functional aspects of the execution. Such preferences can be incorporated in the model easily.

In a multi-level set up, the activities that are re-executed or rolled back would, in general, be composite activities, that too executed by different parties autonomously. Therefore, the choices for re-execution and roll back may be limited and considerable pre-planning may be required in the design phase of the contract.

5 Discussion and Conclusion

In this paper, we have developed a framework for e-contract commitment by considering transactional properties for executions of activities of the e-contract. Accommodating the transactional properties can improve an e-contract design and, in turn, help in the enactment of the underlying contract. Some important aspects are the following.

i. Level-wise definitions of compensatability and retriability clarify the properties and requirements in the executions of activities and sub-activities, in contracts and sub-contracts. This helps in delegating responsibilities for satisfying the required properties in the executions to relevant parties precisely and unambiguously.
ii. Closure of the contract can be tied to closure of the activities and commitments. Features such as "the life of a contract may extend far beyond the termination of execution of the activities" are accommodated fairly easily.
iii. Terms of payments for the activities (including the contract-activity) can be related to the execution states of the activities. (We omit the details for lack of space.)

We believe that our transactional properties will be useful in other applications also, irrespective of whether the activities are electronic, non-electronic or both.

In our multi-level model, we get composite activities in the form of a special type of acyclic graphs. This structure may be sufficient for most activities. It contains sequence, AND splits and joins, and OR splits and joins, for instance. However, the model can be extended to get composite activities in arbitrary acyclic graph structures. We omit the details for lack of space. In our on-going work, we are looking into several issues that arise with the incorporation of our model in the ER^{EC} framework.

References

1. Chiu, D.K.W., Karlapalem, K., Li, Q., Kafeza, E.: Workflow View Based E-Contracts in a Cross-Organizational E-Services Environment. Distributed and Parallel Databases 12(2/3), 193–216 (2002)
2. Chrysanthis, P.K., Ramamritham, K.: A Formalism for Extended Transaction Models. In: Proc. of the 17th Int. Conf. on Very Large Data Bases, pp. 103–112 (1991)
3. Green, P., Vonk, J.: A Taxonomy of Transactional Workflow Support. International Journal of Cooperative Information Systems 15, 87–118 (2006)
4. Grosof, B., Poon, T.: SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies and Process Descriptions. In: Proc. of the 12th WWW Conference (2003)
5. Kafeza, E., Chiu, D., Kafeza, I.: View-based Contracts in an E-service Cross-organizational Workflow Environment. In: Proc. of 2nd Int. Workshop on Technologies for E-service (2001)
6. Koetsier, M., Grefen, P., Vonk, J.: Contract Model, Technical Report, Cross-Organizational/Workflow, Crossflow ESPRITE/28635 (1999)
7. Radha Krishna, P., Karlapalem, K., Chiu, D.K.W.: An EREC Framework for E-Contract Modeling, Enactment and Monitoring. Data and Knowledge Engineering 51, 31–58 (2004)
8. Radha Krishna, P., Karlapalem, K., Dani, A.R.: From Contracts to E-Contracts: Modeling and Enactment. Information Technology and Management Journal 4(1), 363–387 (2005)
9. Schuldt, H., Alonso, G., Beeri, C., Schek, H.J.: Atomicity and Isolation for Transactional Processes. ACM Transactions on Database Systems 27, 63–116 (2002)
10. Vidyasankar, K.: Atomicity of Global Transactions in Distributed Heterogeneous Database Systems. In: Proc. of the DEXA-91, pp. 321–326 (1991)
11. Vidyasankar, K., Vossen, G.: A Multi-Level Model for Web Service Composition. In: Proc. of the 3rd IEEE International Conference on Web Services, San Diego, U.S.A, pp. 462–469. IEEE Computer Society Press, Los Alamitos (2004)
12. Vidyasankar, K., Vossen, G.: Multi-Level Modeling of Web Service Compositions with Transactional Properties, Technical Report, Memorial University, St. John’s, Canada (2007)
13. Wang, T., Grefen, P., Vonk, J.: Abstract Transaction Construct: Building a Transaction Framework for Contract-driven. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 434–439. Springer, Heidelberg (2006)
14. Xu, L.: A Multi-party Contract Model. ACM SIGecom Exchanges 5(1), 13–23 (2004)

Decentralised Commitment for Optimistic Semantic Replication*

Pierre Sutra[1], João Barreto[2], and Marc Shapiro[1]

[1] Université Paris VI and INRIA Rocquencourt, France
[2] INESC-ID and Instituto Superior Técnico, Lisbon, Portugal

Abstract. We study large-scale distributed cooperative systems that use optimistic replication. We represent a system as a graph of actions (operations) connected by edges that reify semantic constraints between actions. Constraint types include conflict, execution order, dependence, and atomicity. The local state is some schedule that conforms to the constraints; because of conflicts, client state is only tentative. For consistency, site schedules should converge; we designed a decentralised, asynchronous commitment protocol. Each client makes a proposal, reflecting its tentative and/or preferred schedules. Our protocol distributes the proposals, which it decomposes into semantically-meaningful units called candidates, and runs an election between comparable candidates. A candidate wins when it receives a majority or a plurality. The protocol is fully asynchronous: each site executes its tentative schedule independently, and determines locally when a candidate has won an election. The committed schedule is as close as possible to the preferences expressed by clients.

1 Introduction

In a large-scale cooperative system, access to shared data is a performance and availability bottleneck. One solution is optimistic replication (OR), where a process may read or update its local replica without synchronising with remote sites [17]. OR decouples data access from network access.

In OR, each site makes progress independently, even while others are slow, currently disconnected, or currently working in isolated mode. OR is well suited to peer-to-peer systems and to devices with occasional connectivity.

Some limited knowledge of semantics provides a lot of extra power and flexibility. Therefore, we model the system as a graph, called a multilog, where each vertex represents an action (i.e., an operation proposed by some client), and an edge is a semantic relation between vertices, called a constraint. Our constraints include conflict, ordered execution, causal dependence, and atomicity. Each site has its own multilog, which contains actions submitted by the local client, and their constraints, as well as those received from other sites. The current state is some execution schedule that contains actions from the site's multilog, arranged to conform with its constraints. For instance, when actions are antagonistic, at least one must abort; an action that depends on an aborted action

* This research is funded in part by the European project Grid4All, the French project Respire and by FCT grant SFRH/BD/13859, Portugal.

must abort too; non-commutative actions should be scheduled in the same order everywhere, etc. The site may choose any conforming schedule, e.g., one that minimises aborts, or one that reflects user preferences.

For consistency, sites should agree on a common, stable and correct schedule. We call this agreement *commitment*. Some cooperative OR systems never commit, such as Roam [16] or Draw-Together [6]. Previous work on commitment for semantic OR such as Bayou [20] or IceCube [15] centralises the agreement at a central site. Other work decentralises commitment (e.g., Paxos consensus [11]) but ignores semantics. It is difficult to reconcile semantics and decentralisation. One possible approach would use Paxos to compute a total order, and abort any actions for which this order would violate a constraint. However this approach aborts actions unnecessarily. Furthermore, the arbitrary total order may be very different from what users expect.

A better approach is to order only non-commuting pairs of actions, to abort only when actions are antagonistic, to minimise dependent aborts, and to remain close to user expectations. We propose an efficient, decentralised protocol that uses semantic information for this purpose. Participating sites make and exchange proposals asynchronously; our algorithm decomposes each one into semantically-meaningful candidates; it runs elections between comparable candidates. A candidate that collects a majority or a plurality wins its election. Voting ensures that the common schedule is similar to the tentative schedules, minimising user surprise. Our protocol orders only non-commuting actions and minimises unnecessary aborts.

This paper makes several contributions:

- Our algorithm combines a number of known techniques in a novel manner.
- We identify the concept of a semantically-meaningful unit for election (which we call a candidate).
- We propose an efficient commitment protocol system that is both decentralised and semantic-oriented, and that has weak communication requirements.
- We show how to minimise user surprise, the committed schedule being similar to local tentative schedules.
- We prove that the protocol is safe even in the presence of non-byzantine faults. The protocol is live as long as a sufficient number of votes are received.

The outline of this paper follows. Section 2 introduces our system model and our vocabulary. Section 3 discusses an abstraction of classical OR approaches that is later re-used in our algorithm. Section 4 specifies client behaviour. Our commitment protocol is specified in Section 5. Section 6 provides a proof outline and adresses message cost. We compare with related work in Section 7. In conclusion, Section 8 discusses our results and future work.

2 System Model

Following the ACF model [18], an OR system is an asynchronous distributed system of n *sites* $i, j, \ldots \in \mathcal{I}$. A site that crashes eventually recovers with its identity and persistent memory intact (but may miss some messages in the interval). Clients propose actions (deterministic operations) noted $\alpha, \beta, \ldots \in A$. An action might request, for instance, "Debit 100 euros from bank account number 12345."

A *multilog* is a quadruple $M = (K, \rightarrow, \lhd, \nparallel)$, representing a graph where the vertices K are actions, and $\rightarrow$, $\lhd$ and $\nparallel$ (pronounced NotAfter, Enables and NonCommuting respectively) are three sets of edges called *constraints*. We will explain their semantics shortly.[1]

We identify a state with a *schedule* S, a sequence of distinct actions ordered by $<_S$ executed from the common initial state INIT. The following safety condition defines semantics of NotAfter and Enables in relation to schedules. We define $\Sigma(M)$, the set of schedules S that are *sound* with respect to multilog M, as follows:

$$S \in \Sigma(M) \overset{\text{def}}{=} \forall \alpha, \beta \in A \quad \begin{cases} \text{INIT} \in S \\ \alpha \in S \Rightarrow \alpha \in K \\ \alpha \in S \wedge \alpha \neq \text{INIT} \Rightarrow \text{INIT} <_S \alpha \\ (\alpha \rightarrow \beta) \wedge \alpha, \beta \in S \Rightarrow \alpha <_S \beta \\ (\alpha \lhd \beta) \Rightarrow (\beta \in S \Rightarrow \alpha \in S) \end{cases}$$

Constraints represent scheduling relations between actions: NotAfter is a (non-transitive) ordering relation and Enables is right-to-left (non-transitive) implication.[2]

Constraints represent semantic relations between actions. For instance, consider a database system (more precisely, a serialisable database that transmits transactions by value, such as DBSM [13]). Assume shared variables x, y, z are initially zero. Two concurrent transactions $T_1 = r(x)0; w(z)1$ and $T_2 = w(x)2$ are related by $T_1 \rightarrow T_2$, since T_1 read a value that precedes T_2's write.[3] T_1 and $T_3 = r(z)0; w(x)3$ are antagonistic, i.e., one or the other (or both) must abort, as each is NotAfter the other. In the execution $T_1; T_4$ where $T_4 = r(z)1$, the latter transaction depends causally on the former, i.e., they may run only in that order, and T_4 aborts if T_1 aborts; we write $T_1 \rightarrow T_4 \wedge T_1 \lhd T_4$. As another example, Section 6.4 discusses how to encode the semantics of database transactions with constraints.

Non-commutativity imposes a liveness obligation: the system must put a NotAfter between non-commuting actions, or abort one of them. (Therefore, non-commutativity does not appear in the above safety condition.) The system also has the obligation to resolve antagonisms by aborting actions.

For instance, transactions T_1 and $T_5 = r(y)0$ commute if x, y and z are independent. In a database system that commits operations (as opposed to commiting values), transactions T_6 ="Credit 66 euros to Account 12345" and T_7 ="Credit 77 euros to Account 12345" commute since addition is a commutative operation, but T_6 and T_8 ="Debit 88 euros from Account 12345" do not, if bank accounts are not allowed to become negative. We write $T_6 \nparallel T_8$.

[1] Multilog union, inclusion, difference, etc., are defined as component-wise union, inclusion, difference, etc., respectively. For instance if $M = (K, \rightarrow, \lhd, \nparallel)$ and $M' = (K', \rightarrow', \lhd', \nparallel')$ their union is $M \cup M' = (K \cup K', \rightarrow \cup \rightarrow', \lhd \cup \lhd', \nparallel \cup \nparallel')$.

[2] A constraint is a relation in $A \times A$. By abuse of notation, for some relation $\mathcal{R}$, we write equivalently $(\alpha \,\mathcal{R}\, \beta) \in M$ or $\alpha \,\mathcal{R}\, \beta$ or $(\alpha, \beta) \in \mathcal{R}$. $\nparallel$ is symmetric and $\lhd$ is reflexive. They do not have any further special properties; in particular, $\rightarrow$ and $\lhd$ are not transitive, are not orders, and may be cyclic.

[3] $r(x)n$ stands for a read of x returning value n, and $w(x)n$ writes the value n into x.

Order, antagonism and non-commutativity are collectively called conflicts.[4]

Clients submit actions to their local site; sites exchange actions and constraints asynchronously. The current knowledge of Site i at time t is the distinguished *site-multilog* $M_i(t)$. Initially, $M_i(0) = (\{\text{INIT}\}, \varnothing, \varnothing, \varnothing)$, and it grows over time, as we will explain later. A site's current state is the *site-schedule* $S_i(t)$, which is some (arbitrary) schedule $\in \Sigma(M_i(t))$.

An action executes tentatively only, because of conflicts and related issues. However, an action might have sufficient constraints that its execution is stable. We distinguish the following interesting subsets of actions relative to M.

- *Guaranteed* actions appear in every schedule of $\Sigma(M)$. Formally, $Guar(M)$ is the smallest subset of K satisfying: $\text{INIT} \in Guar(M) \wedge ((\alpha \in Guar(M) \wedge \beta \lhd \alpha) \Rightarrow \beta \in Guar(M))$.
- *Dead* actions never appear in a schedule of $\Sigma(M)$. $Dead(M)$ is the smallest subset of A satisfying: $((\alpha_1, \ldots, \alpha_{m \geq 0} \in Guar(M)) \wedge (\beta \rightarrow \alpha_1 \rightarrow \ldots \rightarrow \alpha_m \rightarrow \beta) \Rightarrow \beta \in Dead(M)) \wedge ((\alpha \in Dead(M) \wedge \alpha \lhd \beta) \Rightarrow \beta \in Dead(M))$.
- *Serialised* actions are either dead or ordered with respect to all non-commuting constraints. $Serialised(M) \overset{\text{def}}{=} \{\alpha \in K | \forall \beta \in K,\ \alpha \not\parallel \beta \Rightarrow \alpha \rightarrow \beta \vee \beta \rightarrow \alpha \vee \beta \in Dead(M) \vee \alpha \in Dead(M)\}$.
- *Decided* actions are either dead, or both guaranteed and serialised. $Decided(M) \overset{\text{def}}{=} Dead(M) \cup (Guar(M) \cap Serialised(M))$.
- *Stable* (i.e., *durable*) actions are decided, and all actions that precede them by NotAfter or Enables are themselves stable: $Stable(M) \overset{\text{def}}{=} Dead(M) \cup \{\alpha \in Guar(M) \cap Serialised(M) | \forall \beta \in A \quad (\beta \rightarrow \alpha \vee \beta \lhd \alpha) \Rightarrow \beta \in Stable(M)\}$.

To *decide* an action α relative to a multilog M, means to add constraints to the M, such that $\alpha \in Decided(M)$. In particular, to guarantee α, we add $\alpha \lhd \text{INIT}$ to the multilog, and to kill α, we add $\alpha \rightarrow \alpha$; to serialise non-commuting actions α and β, we add either $\alpha \rightarrow \beta$, $\beta \rightarrow \alpha$, $\alpha \rightarrow \alpha$, or $\beta \rightarrow \beta$.

Multilog M is said sound iff $\Sigma(M) \neq \varnothing$, or equivalently, iff $Dead(M) \cap Guar(M) = \varnothing$. An unsound multilog is definitely broken, i.e., no possible schedule can satisfy all the constraints, not even the empty schedule.

Referring to the standard database terminology, a committed action is one that is both stable and guaranteed, and aborted is the same as dead.

The standard correctness condition in OR systems is Eventual Consistency: if clients stop submitting, eventually all sites reach the same state. We extend this definition by not requiring that clients stop, by requiring that all states be correct, and by demanding decision.

Definition 1. *Eventual Consistency. An OR system is eventually consistent iff it satisfies all the following conditions:*

- *Local soundness (safety): Every site-schedule is sound:* $\forall i,t \quad S_i(t) \in \Sigma(M_i(t))$

[4] Some authors suggest to remove conflicts by transforming the actions [19]. We assume that, if such transformations are possible, they have already been applied.

α	$<$	β	$<$	γ	Decision	
		β	$\not\parallel$	γ	(Serialise)	$\beta \rightarrow \gamma$
guar.	$\leftarrow$	β			(Kill β)	$\beta \rightarrow \beta$
dead	$\lhd$	β			(β is dead)	
		β	$\rhd$	γ	(Kill β)	$\beta \rightarrow \beta$
β not dead by above rules					(Guarantee β)	$\beta \lhd \text{INIT}$

Fig. 1. $\mathcal{A}_{\text{Conservative}(<)}$: Applying semantic constraints to a given total order

– *Mergeability (safety): The union of all the site-multilogs over time is sound:*

$$\Sigma\left(\bigcup_{i,t} M_i(t)\right) \neq \varnothing$$

– *Eventual propagation (liveness):* $\forall i, j \in \mathcal{I} \quad \forall t \quad \exists t' : M_i(t) \subseteq M_j(t')$
– *Eventual decision (liveness): Every submitted action is eventually decided:*

$$\forall \alpha \in A \quad \forall i \in \mathcal{I} \quad \forall t \quad \exists t' : K_i(t) \subseteq \mathit{Decided}(M_i(t'))$$

We assume some form of epidemic communication to fulfill Eventual Propagation. A commitment algorithm aims to fulfill the obligations of Eventual Decision. Of course, it must also satisfy the safety requirements.

3 Classical or Commitment Algorithms

Our proposal builds upon existing commitment algorithms for OR systems. Generally, these either are centralised or do not take constraints into account. We note $\mathcal{A}(M)$ some algorithm that offers decisions based on multilog M ; with no loss of generality, we focus on the outcome of $\mathcal{A}$ at a single site. Assuming M is sound, and noting the result $M' = \mathcal{A}(M)$, $\mathcal{A}$ must satisfy these requirements:

– $\mathcal{A}$ extends its input: $M \subseteq M'$.
– $\mathcal{A}$ may not add actions: $K' = K$.
– $\mathcal{A}$ may add constraints, which are restricted to decisions:

$$\begin{aligned} \alpha \rightarrow' \beta &\Rightarrow (\alpha \rightarrow \beta) \vee (\alpha \not\parallel \beta) \vee (\beta = \alpha) \\ \alpha \lhd' \beta &\Rightarrow (\alpha \lhd \beta) \vee (\beta = \text{INIT}) \\ \not\parallel' &= \not\parallel \end{aligned}$$

– M' is sound.
– M' is stable: $\mathit{Stable}(M') = K$.

$\mathcal{A}$ could be any algorithm satisfying the requirements.

One possible algorithm, $\mathcal{A}_{\text{Conservative}(<)}$, first orders actions, then kills actions for which the order is unsafe. It proceeds as follows (see Figure 1). Let $<$ be a total order of actions and M a sound multilog. The algorithm decides one action at time, varying over all actions, left to right; call the current action β. Consider actions α and γ such that $\alpha < \beta < \gamma$: α has already been decided, and γ has not. If $\beta \not\parallel \gamma$, then serialise them in schedule order. If $\beta \rightarrow \alpha$, and α is guaranteed, kill β, because the schedule and the constraint are incompatible. If $\gamma \lhd \beta$, conservatively kill β, because it is not known whether γ can be guaranteed. By definition, if $\alpha \lhd \beta$ and α is dead, then β is dead. If β is not dead by any of the above rules, then decide β guaranteed (by adding $\beta \lhd \text{INIT}$ to the multilog). The resulting $\Sigma(\mathcal{A}_{\text{Conservative}(<)}(M))$ contains a unique schedule.

It should be clear that this approach is safe but tends to kill actions unnecessarily.

The Bayou system [20] applies $\mathcal{A}_{\text{Conservative}(<)}$, where $<$ is the order in which actions are received at a single primary site. An action aborts if it fails an application-specific precondition, which we reify as a $\rightarrow$ constraint.

In the Last-Writer-Wins (LWW) approach [7], an action (completely overwriting some datum) is stamped with the time it is submitted. Two actions that modify the same datum are related by $\rightarrow$ in timestamp order. Sites execute actions in arbitrary order and apply $\mathcal{A}_{\text{Conservative}(<)}$. Consequently, a datum has the state of the most recent write (in timestamp order).

The decisions computed by the above systems are mostly arbitrary. A better way would be to minimise aborts, or to follow user preferences, or both. This was the approach of the IceCube system [15]. $\mathcal{A}_{\text{IceCube}}$ is an optimization algorithm that minimises the number of dead actions in $\mathcal{A}_{\text{IceCube}}(M)$. It does so by heuristically comparing all possible sound schedules that can be generated from the current site-multilog. The system suggests a number of possible decisions to the user, who states his preference.

Except for LWW, which is decentralised but deterministic, the above algorithms centralise commitment at a primary site.

To decentralise decision, one approach might be to determine a global total order $<$, using a decentralised consensus algorithm such as Paxos [11], and apply $\mathcal{A}_{\text{Conservative}(<)}$. As above, this order is arbitrary and $\mathcal{A}_{\text{Conservative}(<)}$ tends to kill unnecessary. Instead, our algorithm allows each site to propose decisions that minimises aborts and follows local client preferences, and to reach consensus on these proposals in a decentralised manner. This is the subject of the rest of this paper.

4 Client Operation

We now begin the discussion of our algorithm. We start with a specification of client behaviour.

4.1 Client Behaviour and Client Interaction

An application performs tentative operations by submitting actions and constraints to its local site-multilog; they will eventually propagate to all sites.

We abstract application semantics by postulating that clients have access to a sound multilog containing all the semantic constraints: $\mathcal{M} = (A, \rightarrow_{\mathcal{M}}, \lhd_{\mathcal{M}}, \not\parallel_{\mathcal{M}})$. For an example $\mathcal{M}$, see Section 6.4.

Algorithm 1. *ClientActionsConstraints*(L)

```
Require: $L \subseteq A$
1: $K_i := K_i \cup L$
2: for all $(\alpha, \beta) \in K_i \times K_i$ such that $\alpha \rightarrow_{\mathcal{M}} \beta$ do
3:     $\rightarrow_i := \rightarrow_i \cup \{(\alpha, \beta)\}$
4: for all $(\alpha, \beta) \in K_i \times K_i$ such that $\alpha \lhd_{\mathcal{M}} \beta$ do
5:     $\lhd_i := \lhd_i \cup \{(\alpha, \beta)\}$
6: for all $(\alpha, \beta) \in K_i \times K_i$ such that $\alpha \nparallel_{\mathcal{M}} \beta$ do
7:     $\nparallel_i := \nparallel_i \cup \{(\alpha, \beta)\}$
```

As the client submits actions L to the site-multilog, function *ClientActionsConstraints* (Algorithm 1) adds constraints with respect to actions that the site already knows.[5]

To illustrate, consider Alice and Bob working together. Alice uses their shared calendar at Site 1, and Bob at Site 2. Planning a meeting with Bob in Paris, Alice submits two actions: $\alpha =$"Buy train ticket to Paris next Monday at 10:00" and $\beta =$"Attend meeting". As β depends causally on α, $\mathcal{M}$ contains $\alpha \rightarrow_{\mathcal{M}} \beta \wedge \alpha \lhd_{\mathcal{M}} \beta$. Alice calls *ClientActionsConstraints*$(\{\alpha\})$ to add action α to site-multilog M_1, and, some time later, similarly for β. At this point, Algorithm 1 adds the constraints $\alpha \rightarrow \beta$ and $\alpha \lhd \beta$ taken from $\mathcal{M}$.

4.2 Multilog Propagation

When a client adds new actions L into a site-multilog, L and the constraints computed by *ClientActionsConstraints*, form a multilog that is sent to remote sites. Upon reception, receivers merge this multilog into their own site-multilog. By this so-called epidemic communication [3], every site eventually receive all actions and constraints submitted at any site.

When Site i receives a multilog M, it executes function *ReceiveAndCompare* (Algorithm 2), which first merges what it received into the local site-multilog. Then, if any conflicts exist between previously-known actions and the received ones, it adds the corresponding constraints to the site-multilog.[6]

Let us return to Alice and Bob. Suppose that Bob now adds action γ, meaning "Cancel the meeting," to M_2. Action γ is antagonistic with action β; hence, $\beta \rightarrow_{\mathcal{M}} \gamma \wedge \gamma \rightarrow_{\mathcal{M}} \beta$. Some time later, Site 2 sends its site-multilog to Site 1; when Site 1 receives it, it runs Algorithm 2, notices the antagonism, and adds constraint $\beta \rightarrow \gamma \wedge \gamma \rightarrow \beta$ to M_1. Thereafter, site-schedules at Site 1 may include either β or γ, but not both.

[5] In the pseudo-code, we leave the current time t implicit. A double-slash and sans-serif font indicates a comment, as in // This is a comment.

[6] *ClientActionsConstraints* provides constraints between successive actions submitted at the same site. These consist typically of dependence and atomicity constraints. In contrast, *ReceiveAndCompare* computes constraints between independently-submitted actions.

Algorithm 2. *ReceiveAndCompare*(M)

```
Declare: $M = (K, \rightarrow, \lhd, \nparallel)$ a multilog receives from a remote site
  $M_i := M_i \cup M$
  for all $(\alpha, \beta) \in K_i \times K_i$ such that $\alpha \rightarrow_{\mathcal{M}} \beta$ do
      $\rightarrow_i := \rightarrow_i \cup \{(\alpha, \beta)\}$
  for all $(\alpha, \beta) \in K_i \times K_i$ such that $\alpha \nparallel_{\mathcal{M}} \beta$ do
      $\nparallel_i := \nparallel_i \cup \{(\alpha, \beta)\}$
```

5 A Decentralised Commitment Protocol

Epidemic communication ensures that all site-multilogs eventually receive all information, but site-schedules might still differ between sites.

For instance, let us return to Alice and Bob. Assuming users add no more actions, eventually all site-multilogs become $(\{\text{INIT}, \alpha, \beta, \gamma\}, \{\alpha \rightarrow \beta, \beta \rightarrow \gamma, \gamma \rightarrow \beta\}, \{\alpha \lhd \beta\}, \varnothing)$. In this state, actions remain tentative; at time t, Site 1 might execute $S_1(t) = \text{INIT}; \alpha; \beta$, Site 2 $S_2(t) = \text{INIT}; \alpha; \gamma$, and just INIT at $t+1$. A commitment protocol ensures that α, β and γ eventually stabilise, and that both Alice and Bob learn the same outcome. For instance, the protocol might add $\beta \lhd \text{INIT}$ to M_1, which guarantees β, thereby both guaranteeing α and killing γ. α, β and γ are now decided and stable at Site 1. M_1 eventually propagates to other sites; and inevitably, all site-schedules eventually start with $\text{INIT}; \alpha; \beta$, and γ is dead everywhere.

5.1 Overview

Our key insight is that eventual consistency is equivalent to the property that the site-multilogs of all sites share a common *well-formed prefix* (defined hereafter) of stable actions, which grows to include every action eventually. Commitment serves to agree on an extension of this prefix. As clients continue to make optimistic progress beyond this prefix, the commitment protocol can run asynchronously in the background.

In our protocol, different sites run instances of $\mathcal{A}$ to make proposals; a proposal being a tentative well-formed prefix of its site-multilog. Sites agree via a decentralised election. This works even if $\mathcal{A}$ is non-deterministic, or if sites use different $\mathcal{A}$ algorithms. We recommend IceCube [15] but any algorithm satisfying the requirements of Section 3 is suitable.

In what follows, i represents the current site, and j, k range over $\mathcal{J}$.

We distinguish two roles at each site, proposers and acceptors. Each proposer has a fixed *weight*, such that $\sum_{k \in \mathcal{J}} weight_k = 1$. In practice, we expect only a small number of sites to have non-zero weights (in the limit one site might have weight 1, this is a primary site as in Section 3), but the safety of our protocol does not depend on how weights are allocated. To simplify exposition, weights are distributed ahead of time and do not change; it is relatively straightforward to extend the current algorithm, allowing weights to vary between successive elections.

An acceptor at some site computes the outcome of an election, and inserts the corresponding decision constraints into the local site-multilog.

Each site stores the most recent proposal received from each proposer in array $proposals_i$, of size n (the number of sites). To keep track of proposals, each entry $proposals_i[k]$ carries a logical timestamp, noted $proposals_i[k].ts$. Timestamping ensures the liveness of the election process despite since links between nodes are not necessarily FIFO.

Algorithm 3. Algorithm at Site i

```
Declare: $M_i$: local site-multilog
Declare: $proposals_i[n]$: array of proposals, indexed by site; a proposal is a multilog
1: $M_i := (\{\text{INIT}\}, \varnothing, \varnothing, \varnothing)$
2: $proposals_i := [((\{\text{INIT}\}, \varnothing, \varnothing, \varnothing), 0), \ldots, ((\{\text{INIT}\}, \varnothing, \varnothing, \varnothing), 0)]$
3: loop // Epidemic transmission
4:     Choose $j \neq i$;
5:     Send copy of $M_i$ and $proposals_i$ to $j$
6: ||
7: loop // Epidemic reception
8:     Receive multilog $M$ and proposals $P$ from some site $j \neq i$
9:     $ReceiveAndCompare(M)$ // Compute conflict constraints
10:    $MergeProposals(P)$
11: ||
12: loop // Client submits
13:    Choose $L \subseteq A$
14:    $ClientActionsConstraints(L)$ // Submit actions, compute local constraints
15: ||
16: loop // Compute current local state
17:    Choose $S_i \in \Sigma(M_i)$
18:    Execute $S_i$
19: ||
20: loop // Proposer
21:    $UpdateProposal$ // Suppress redundant parts
22:    $proposals_i[i] := \mathcal{A}(M_i \cup proposals_i[i])$ // New proposal, keeping previous
23:    Increment $proposals_i[i].ts$
24: ||
25: loop // Acceptor
26:    $Elect$
```

Each site performs Algorithm 3. First it initialises the site-multilog and proposals data structures, then it consists of a number of parallel iterative threads, detailed in the next sections. Within a thread, an iteration is atomic. Iterations are separated by arbitrary amounts of time.

5.2 Epidemic Communication

The first two threads (lines 3–10) exchange multilogs and proposals between sites. Function *ReceiveAndCompare* (defined in Algorithm 2, Section 4.2) compares actions

Algorithm 4. *UpdateProposal*

```
1: Let $P = (K_P, \rightarrow_P, \lhd_P, \nparallel_P) = proposals_i[i]$
2: $K_P := K_P \setminus Decided(M_i)$
3: $\rightarrow_P := \rightarrow_P \cap K_P \times K_P$
4: $\lhd_P := \lhd_P \cap K_P \times K_P$
5: $\nparallel_P := \varnothing$
6: $proposals_i[i] := P$
```

newly received to already-known ones, in order to compute conflict constraints. In Algorithm 6 a receiver updates its own set of proposals with any more recent ones.

5.3 Client, Local State, Proposer

The third thread (lines 12–14) constitutes one half of the client. An application submits tentative operations to its local site-multilog, which the site-schedule will (hopefully) execute in the fourth thread. Constraints relating new actions to previous ones are included at this stage by function *ClientActionsConstraints* (defined in Algorithm 1).

The other half of the client is function *ReceiveAndCompare* (Algorithm 2) invoked in the second thread (line 9).

The fourth thread (lines 16–18) computes the current tentative state by executing some sound site-schedule.

The fifth thread (20–23) computes proposals by invoking $\mathcal{A}$. A proposal extends the current site-multilog with proposed decisions. A proposer may not retract a proposal that was already received by some other site. Passing argument $M_i \cup proposals_i[i]$ to $\mathcal{A}$ ensures that these two conditions are satisfied.

However, once a candidate has either won or lost an election, it becomes redundant; *UpdateProposal* removes it from the proposal (Algorithm 4).

The last thread is described in the next section.

5.4 Election

The last thread (25–26) conducts elections. Several elections may be taking place at any point in time. An acceptor is capable of determining locally the outcome of elections. A proposal can be decomposed into a set of eligible candidates.

Eligible candidates. A candidate cannot be just any subset of a proposal. Consider, for instance, proposal $P = (\{\text{INIT}, \alpha, \gamma\}, \{\alpha \rightarrow \gamma, \gamma \rightarrow \alpha, \alpha \rightarrow \alpha\}, \{\gamma \lhd \text{INIT}\}, \varnothing)$, and some candidate X extracted from P. If X could contain γ and not α, then we might guarantee γ without killing α, which would be incorrect. According to this intuition, X must be a *well-formed prefix* of P:

Definition 2. *Well-formed prefix. Let $M = (K, \rightarrow, \lhd, \nparallel)$ and $M' = (K', \rightarrow', \lhd', \nparallel')$ be two multilogs. M' is a* well-formed prefix *of M, noted $M' \overset{wf}{\sqsubseteq} M$, if (i) it is a subset of M, (ii) it is stable, (iii) it is left-closed for its actions, and (iv) it is closed for its constraints.*

$$M' \overset{wf}{\sqsubset} M \overset{\text{def}}{=} \begin{cases} M' \subseteq M \\ K' = \mathit{Stable}(M') \\ \forall \alpha, \beta \in A \quad \beta \in K' \Rightarrow \begin{cases} \alpha \rightarrow \beta \Rightarrow \alpha \rightarrow' \beta \\ \alpha \lhd \beta \Rightarrow \alpha \lhd' \beta \\ \alpha \not\parallel \beta \Rightarrow \alpha \not\parallel' \beta \end{cases} \\ \forall \alpha, \beta \in A \quad (\alpha \rightarrow' \beta \vee \alpha \lhd' \beta \vee \alpha \not\parallel' \beta) \Rightarrow \alpha, \beta \in K' \end{cases}$$

A well-formed prefix is a semantically-meaningful unit of proposal. For instance, if a $\rightarrow$ or $\lhd$ cycle is present in M , every well-formed prefix either includes the whole cycle, or none of its actions.

Unfortunately, because of concurrency and asynchronous communication, it is possible that some sites know of a $\rightarrow$ cycle and not others; or more embarassingly, that sites know only parts of a cycle. Therefore we also require the following property:

Definition 3. *Eligible candidates. An action is* eligible *in set L if all its predecessors by client NotAfter, Enables and NonCommuting relations are in L. A candidate multilog M is eligible if all actions in K are eligible in K:* $\mathit{eligible}(M) \overset{\text{def}}{=} \forall \alpha, \beta \in A \times K \quad (\alpha \rightarrow_{\mathcal{M}} \beta \vee \alpha \not\parallel_{\mathcal{M}} \beta \vee \alpha \lhd_{\mathcal{M}} \beta) \Rightarrow \alpha \in K.$

To compute eligibility precisely would require local access to the distributed state, which is impossible. Therefore acceptors must compute a safe approximation (i.e., false negatives are allowed) of eligibility. For instance, in the database example, a sufficient condition for transaction T to be eligible at Site i is that all transactions submitted (at any site) concurrently with T are also known at Site i. Indeed, all such transactions have gone through either *ClientActionsConstraints* or *ReceiveAndCompare*; hence according to Table 1, T is eligible.

Computation of votes. We define a vote as a pair $(\mathit{weight}, \mathit{siteId})$. The comparison operator for votes breaks ties by comparing site identifiers: $(w,i) > (w',i') \overset{\text{def}}{=} w > w' \vee (w = w' \wedge i > i')$. Therefore, votes add up as follows: $(w,i) + (w',i') \overset{\text{def}}{=} (w + w', \max(i,i'))$. Candidates are *compatible* if their union is sound: $\mathit{compatible}(M,M') \overset{\text{def}}{=} \Sigma(M \cup M') \neq \varnothing$. The votes of compatible candidates add up; $\mathit{tally}(X)$ computes the total vote for some candidate X:

$$\mathit{tally}(X) \overset{\text{def}}{=} \sum_{k : X \overset{wf}{\sqsubset} \mathit{proposals}_i[k]} (\mathit{weight}_k, k)$$

An election pits some candidate against *comparable* candidates from all other sites. Two multilogs are comparable if they contain the same set of actions: $\mathit{comparable}(M,M') \overset{\text{def}}{=} K = K'$. The direct opponents of candidate X in some election are comparable candidates that X does not prefix:

$$\mathit{opponents}(X) \overset{\text{def}}{=} \{B | \exists k : B \overset{wf}{\sqsubset} \mathit{proposals}_i[k] \quad \wedge \mathit{comparable}(B,X) \wedge X \overset{wf}{\not\sqsubset} B)\}$$

However, we must also count missing votes, i.e., the weights of sites whose proposals do not yet include all actions in X. Function $\mathit{cotally}(X)$ adds these up:

$$\mathit{cotally}(X) \overset{\text{def}}{=} \sum_{k : K_X \not\subseteq K_{\mathit{proposals}_i[k]}} (\mathit{weight}_k, k)$$

Algorithm 5. *Elect*

```
1: Let $X$ be a multilog such that:
         $\exists k \in \mathcal{J} : X \overset{wf}{\sqsubseteq} proposals_i[k]$
       $\wedge\ X \not\sqsubseteq M_i$
       $\wedge\ eligible(X)$
       $\wedge\ tally(X) > \max_{B \in opponents(X)} (tally(B)) + cotally(X)$
2: if such an $X$ exists then
3:     Choose such an $X$
4:     $M_i := M_i \cup X$
```

Algorithm 6. *MergeProposals*(P)

```
1: for all $k$ do
2:     if $proposals_i[k].ts < P[k].ts$ then
3:         $proposals_i[k] := P[k]$
4:         $proposals_i[k].ts := P[k].ts$
```

Algorithm 5 depicts the election algorithm. A candidate is a well-formed prefix of some proposal. We ignore already-elected candidates and we only consider eligible ones. A candidate wins its election if its tally is greater than the tally of any direct opponent, plus its cotally. Note that, as proposals are received, cotally tends towards 0, therefore some candidate is eventually elected. We merge the winner into the site-multilog.

5.5 Example

We return to our example. Recall that, once Alice and Bob have submitted their actions, and Site 1 and Site 2 have exchanged site-multilogs, both site-multilogs are equal to $(\{\text{INIT}, \alpha, \beta\}, \{\alpha \rightarrow \beta, \alpha \rightarrow \gamma, \gamma \rightarrow \alpha\}, \{\alpha \lhd \beta\}, \varnothing)$. Now Alice (Site 1) proposes to guarantee α and β, and to kill γ: $proposals_1[1] = M_1 \cup \{\beta \lhd \text{INIT}\}$. In the meanwhile, Bob at Site 2 proposes to guarantee γ and α, and to kill β: $proposals_2[2] = M_2 \cup \{\gamma \lhd \text{INIT}, \alpha \lhd \text{INIT}\}$. These proposals are incompatible; therefore that the commitment protocol will eventually agree on at most one of them.

Consider now a third site, Site 3; assume that the three sites have equal weight $\frac{1}{3}$. Imagine that Site 3 receives Site 2's site-multilog and proposal, and sends its own proposal that is identical to Site 1's. Sometime later, Site 3 sends its proposal to Site 1. At this point, Site 1 has received all sites' proposals. Now Site 1 might run an election, considering a candidate X equal to $proposals_1[1]$. X is indeed a well-formed prefix of $proposals_1[1]$; now suppose that X is eligible as all sites have voted on K_X; $tally(X) = \frac{2}{3}$ is greater than that of X's only opponent ($tally(proposals_1[2]) = \frac{1}{3}$); and $cotally(X) = 0$.

Therefore, Site 1 elects X and merges X into M_1. Any other site will either elect X (or some compatible candidate) or become aware of its election by epidemic transmission of M_1.

6 Discussion

6.1 Safety Proof Outline

Section 1 states our safety property, the conjunction of mergeability and local soundness. Clearly Algorithm 3 satisfies local soundness; see lines 16–18. We now outline a proof of mergeability.

We say that candidate X is *elected* in a run r at time t, if some acceptor i executes Algorithm 5 in r at t, and elects a candidate Y such that $X \overset{wf}{\sqsubseteq} Y$. Given a run r of Algorithm 3, we note $Elected(r,t)$ the set of candidates elected in r up to time t (inclusive), and $Elected(r)$ the set of candidates elected during r. Observe that, since $\mathcal{M}$ is sound, Algorithm 3 satisfies mergeability in a run r if and only if the acceptors elect a sound set of candidates during r ($\bigcup_{X \in Elected(r)} X$ is sound).

Suppose, by contradiction, that during run r, this set is unsound. As $\mathcal{M}$is sound, by $\mathcal{A}$ candidates are sound. Consequently there must exist an unsound set of candidates $C \subseteq Elected(r)$. Let us now consider the following property:

Definition 4. *Minimality. A multilog M is said minimal iff: $\forall M' \subseteq M \quad M' \overset{wf}{\sqsubseteq} M \Rightarrow M' = M$.*

As candidates are eligible, there must exist two candidates X and X' in C such that: (i) X and X' are non-compatible, and (ii) X and X' are minimal.

We define the following notation. Let i (resp. i') be the acceptor that elects X (resp. X') in r. t is the time where i elects X in r (resp. t' for X' on i'). For a proposer k, t_k (resp. t'_k) is the time at which it sent $proposals_i[k](t)$ to i (resp. $proposals_{i'}[k](t')$ to i'). Q (resp. Q') is the set of proposers that vote for X at t on i (resp. for X' at t' on i'); formally $Q = \{k | X \overset{wf}{\sqsubseteq} proposals_i[k](t)\}$ and $Q' = \{k | X' \overset{wf}{\sqsubseteq} proposals_{i'}[k](t')\}$.

Hereafter, and without loss of generality, we suppose that: (i) $t < t'$, (ii) X is the first candidate non-compatible with X' elected in r, and (iii) $Elected(r, t'-1)$ is sound.

Since i' elects X' at t', at that time on Site i':

$$tally(X') > \max_{B \in opponents(X')} (tally(B)) + cotally(X') \tag{1}$$

Equation 1 defines an upper bound for $tally(X)$ on i at t, as follows. Consider some $k \in Q$. If $t_k < t'_k$ then from Algorithm 4, and the fact that $Elected(r, t'-1)$ is sound, we know that $X \overset{wf}{\sqsubseteq} proposals_{i'}[k](t')$.

If now $t_k > t'_k$, then as $tally(X')$, $opponents(X')$ and $cotally(X')$ define a partition of $\mathcal{J}$, either:

1. k has not yet voted on $K_{X'}$ at t' on i' and its weight is counted in $cotally(X')$.
2. Or, if its vote already includes $K_{X'}$, it is counted in $opponents(X')$ as X is the first candidate non-compatible with X' elected in r, $X \overset{wf}{\sqsubseteq} proposals_i[k](t)$, and $\neg\, compatible(X, X')$.

From these reasonnings (if $t_k < t'_k$ and if $t'_k < t_k$), and Equation 1, we derive:

$$tally_{i'}(X')(t') > tally_i(X)(t) \tag{2}$$

where $tally_k(Z)(\tau)$ means the value of $tally(Z)$ computed at time τ on Site k.

Now consider some $k \in Q'$.

If $t_k > t'_k$ then X being the first candidate non-compatible with X' elected in r, from Algorithm 4, we have $X' \overset{wf}{\sqsubset} proposals_i[k](t)$.

If $t_k < t'_k$, now either

1. $X' \overset{wf}{\sqsubset} proposals_i[k](t)$
2. or k has not yet voted on $X.K$ on i at t.

The reasoning here is similar to $k \in Q$: we use the minimality of X and X', the fact that they are non-compatible, and that X is the first candidate non-compatible with X' elected in r.

From the above, it follows that:

$$tally_{i'}(X')(t') < tally_i(X')(t) + cotally_i(X)(t) \tag{3}$$

Now, combining equations 2 and 3, we conclude that, at site i at time t:

$$tally(X) < \max_{B \in opponents(X)} (tally(B)) + cotally(X) \tag{4}$$

X cannot be elected on i at t. Contradiction.

6.2 Message Cost

Interestingly, the message cost of our protocol varies with application semantics, along two dimensions.

First, the *degree of semantic complexity*, i.e., the complexity of the client constraint graph $\mathcal{M}$, influences the number of votes required. To illustrate, consider an application where all actions are mutually independent, i.e., $\mathcal{M}$contains no constraints. Then, all actions commute with one another, and no action never needs to be killed. Every candidate is trivially eligible, and trivially compatible with all other candidates.

Second, call *degree of optimism d* the size of a batch, i.e., the number of actions that a site may execute tentatively before requiring commitment. This measures both that replicas relax consistency and that clients propose to the same replica, concurrent commutative actions. It takes a chain of $\frac{n}{2}$ messages to construct a majority. A candidates may contain up to d actions. Therefore, the amortised message cost to commit an action is $\frac{n}{2} \times \frac{1}{d}$.

A more detailed evaluation of message cost is left for future work.

6.3 Implementation Considerations

Our pseudo-code was written for clarity, not efficiency. Many optimisations are possible. For instance, a site i does not need to send the whole $proposals_i$ [i]. When sending to j, it suffices to send the difference $proposals_i[i] \setminus proposals_i[j]$.

Conceptually, a multilog grows without bound. However, a stable action, and all its constraints, can safely be deleted.

Table 1. $\mathcal{M}_{\text{SER-DB-after}}$: Constraints for a serialisable database that transmits after-values

	$T \prec T'$	$T \parallel T'$	$T' \prec T$
$RS(T) \cap WS(T') \neq \varnothing$	$T \rightarrow T'$	$T \rightarrow T'$	$T' \rightarrow T \wedge T' \lhd T$
$WS(T) \cap WS(T') \neq \varnothing$	$T \rightarrow T'$	$T \not\parallel T'$	$T' \rightarrow T$

Conceptually, our algorithm executes all actions everywhere. A practical implementation only needs to achieve an equivalent state; in particular actions that do not have side-effects do not have to be replayed. For instance, in a database application, read operations do not to be replayed.[7]

6.4 Example Application

We illustrate the application of our algorithm to a replicated database. The semantic constraints between two transactions depend on several factors: (i) Whether the transactions are related by happens-before or are concurrent. (ii) Whether their read- and write-sets intersect or not. (iii) What consistency criterion is being enforced (for instance, constraints differ between serializability and snapshot isolation [2]). (iv) How, after executing a transaction on some initial site, the system replicates its effects at a remote site: by replaying the transaction, or by applying the after-values computed at the initial site.

Table 1 exhibits semantic constraints between transactions, where (a) the system replicates a transaction by writing its after-values, and (b) transactions are strictly serialisable.[8] Supporting a different semantics, e.g., (a') replaying actions, or (b') SI, requires only some small changes to the table.

7 Related Work

In previous OR systems, commitment was often either centralised at a primary site [15,20] or oblivious of semantics [7,17]. It is very difficult to combine decentralisation with semantics.

Our election algorithm is inspired by Keleher's Deno system [8], a pessimistic system, which performs a discrete sequence of elections. Keleher proposes plurality voting to ensure progress when none of multiple competing proposals gains a majority. The VVWV protocol of Barreto and Ferreira generalizes Deno's voting procedure, enabling continuous voting [1].

The only semantics supported by Deno or VVWV is to enforce Lamport's happens-before relation [10]; all actions are assumed be mutually non-commuting. Happens-before captures potential causality; however an event may happen-before another even

[7] Formally, we need to generalise the equivalence relation between schedules, which currently is based only on $\not\parallel$ [18]. The definition of consistency now becomes that every pair of sites eventually converges to schedules that are equivalent according to the new relation.

[8] $T \prec T'$ denotes T happens-before T [10]. $T \parallel T'$ denotes concurrency, i.e., neither $T \prec T'$, nor $T' \prec T$. $RS(T)$ and $WS(T)$ denote T's read set and write set respectively.

if they are not truly dependent. This paper further generalizes VVWV by considering semantic constraints.

Holliday et al. depict a family of epidemic algorithms to ensure serializability in replicated datbase systems [5]. The three algorithms consider that concurrent conflicting transactions are antagonistic. Two of them abort concurrent conflicting transactions, and the last one (quorum-based) can only commit one transactions among a set of concurrent conflicting ones. Our algorithm consider that concurrent conflicting transactions are not necessarily antagonistic, it tries to optimize the number of committed transactions, computing a best-effort proposal , and electing them with plurality.

ESDS [4] is a decentralised replication protocol that supports some semantics. It allows users to create an arbitrary causal dependence graph between actions. ESDS eventually computes a global total order among actions, but also includes an optimisation for the case where some action pairs commute. ESDS does not consider atomicity or antagonism relations, nor does it consider dead actions.

Bayou [20] supports arbitrary application semantics. User-supplied code controls whether an action is committed or aborted. However the system imposes an arbitrary total execution order. Bayou centralises decision at a single primary replica.

IceCube [9] introduced the idea of reifying semantics with constraints. The IceCube algorithm computes optimal proposals, minimizing the number of dead actions. Like Bayou, commitment in IceCube is centralised at a primary. Compared to this article, IceCube supports a richer constraint vocabulary, which is useful for applications, but harder to reason about formally.

The Paxos distributed protocol [11] computes a total order. Such total order may be used to implement *state-machine replication* [10], whereby all sites execute exactly the same schedule. Such a total order over all actions is necessary only if all actions are mutually non-commuting. In Section 3 we showed how to combine semantic constraints with a total order, but this approach is clearly sub-optimal. Howover, Paxos remains live even if $f < \frac{n}{2}$ sites crash forever, whereas the other systems described here (including ours) block if a site crashes forever. We assume that a site stores its multilogs and its proposals in persistent memory, and that after a crash it with its identity and persistent store intact. This is a fairly reasonable assumption in a well-managed cooperative system. (For instance, each site might actually be implemented as a cluster on a LAN, with redundant storage, and strong consistency internally.)

Generalized Paxos [12] and Generic Broadcast [14] take commutativity relations into account and compute a partial order. They do not consider any other semantic relations. Both Generalized Paxos [12] and our algorithm make progress when a majority is not reached, although through different means. Generalized Paxos starts a new election instance, whereas our algorithm waits for a plurality decision.

8 Conclusion and Future Work

The focus of our study is cooperative applications with rich semantics. Previous approaches to replication did not support a sufficiently rich repertoire of semantics, or relied on a centralized point of commitment. They often impose a total order, which is stronger than necessary.

In contrast, we propose a decentralized commitment protocol for semantically-rich systems. Our approach is to reify semantic relations as constraints, which restrict the scheduling behavior of the system. According to our formal definition of consistency, the system has an obligation to resolve conflicts, and to eventually execute equivalent stable schedules at all sites.

Our protocol is safe in the absence of Byzantine faults, and live in the absence of crashes. It uses voting to avoid any centralization bottleneck, and to ensure that the result is similar to local proposals. It uses plurality voting to make progress even when an election does not reach a majority.

There is an interesting trade-off in the proposal/voting procedure. The system might decide frequently, in small increments, so that users quickly know whether their tentative actions are accepted or rejected. However this might be non-optimal as it may cut off interesting future behaviors. Or it may base its decisions on a large batch of tentative actions, deciding less frequently. This imposes more uncertainty on users, but decisions may be closer to the optimum. We plan to study this trade-off in our future work.

Another future direction is partial replication. In such a system, a site receives only the actions relative to the objects it replicates (and their constraints). A site votes only on the actions it knows. Because constraints might relate actions known only by distinct sites, these sites must agree together; however we expect that global agreement is rarely necessary. By exploiting knowledge of semantic constraints, we hope to limit the scope of a commitment protocol to small-scale agreements, instead of a global consensus.

References

1. Barreto, J., Ferreira, P.: An efficient and fault-tolerant update commitment protocol for weakly connected replicas. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 1059–1068. Springer, Heidelberg (2005)
2. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading (1987)
3. Demers, A.J., Greene, D.H., Hauser, C., Irish, W., Larson, J.: Epidemic algorithms for replicated database maintenance. In: Symp. on Principles of Dist. Comp. (PODC), pp. 1–12, Vancouver, BC, Canada (August 1987). Also appears Op. Sys. Review 22(1), 8–32 (1988)
4. Fekete, A., Gupta, D., Luchangco, V., Lynch, N., Shvartsman, A.: Eventually-serializable data services. Theoretical Computer Science 220(Special issue on Distributed Algorithms), 113–156 (1999)
5. Holliday, J., Steinke, R., Agrawal, D., Abbadi, A.E.: Epidemic algorithms for replicated databases. IEEE Transactions on Knowledge and Data Engineering 15(5), 1218–1238 (2003)
6. Ignat, C.-L., Norrie, M.C.: Draw-Together: Graphical editor for collaborative drawing. In: CSCW. Int. Conf. on Computer-Supported Cooperative Work, Banff, Alberta, Canada, pp. 269–278 (November 2006)
7. Johnson, P.R., Thomas, R.H.: The maintenance of duplicate databases. Internet Request for Comments RFC 677, Information Sciences Institute (January 1976)
8. Keleher, P.J.: Decentralized replicated-object protocols. In: Symp. on Principles of Dist. Comp. (PODC), Atlanta, GA, USA, pp. 143–151. ACM Press, New York (1999)
9. Kermarrec, A.-M., Rowstron, A., Shapiro, M., Druschel, P.: The IceCube approach to the reconciliation of divergent replicas. In: Symp. on Principles of Dist. Comp. (PODC), Newport, RI, USA ACM SIGACT-SIGOPS, ACM Press, New York (2001)

10. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7), 558–565 (1978)
11. Lamport, L.: The part-time parliament. ACM Transactions on Computer Systems 16(2), 133–169 (1998)
12. Lamport, L.: Generalized consensus and Paxos. Technical Report MSR-TR-2005-33, Microsoft Research (March 2005)
13. Pedone, F., Guerraoui, R., Schiper, A.: The database state machine approach. J. of Dist. and Parallel Databases and Technology 14(1), 71–98 (2003)
14. Pedone, F., Schiper, A.: Handling message semantics with generic broadcast protocols. Distributed Computing Journal 15(2), 97–107 (2002)
15. Preguiça, N., Shapiro, M., Matheson, C.: Semantics-based reconciliation for collaborative and mobile environments. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 38–55. Springer, Heidelberg (2003)
16. Ratner, D., Reiher, P., Popek, G.: Roam: A scalable replication system for mobile computing. In: Int. W. on Database & Expert Systems Apps (DEXA), pp. 96–104. IEEE Comp. Society, Los Alamitos, CA, USA (1999)
17. Saito, Y., Shapiro, M.: Optimistic replication. Computing Surveys 37(1), 42–81 (2005)
18. Shapiro, M., Bhargavan, K.: The Actions-Constraints approach to replication: Definitions and proofs. Technical Report MSR-TR-2004-14, Microsoft Research (March 2004)
19. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. Trans. on Comp.-Human Interaction 5(1), 63–108 (1998)
20. Terry, D.B., Theimer, M.M., Petersen, K., Demers, A.J., Spreitzer, M.J., Hauser, C.H.: Managing update conflicts in Bayou, a weakly connected replicated storage system. In: 15th Symp. on Op. Sys. Principles (SOSP), Copper Mountain, CO, USA, pp. 172–182 ACM SIGACT-SIGOPS, ACM Press, New York (1995)

Coordinate BPEL Scopes and Processes by Extending the WS-Business Activity Framework

Stefan Pottinger[2], Ralph Mietzner[1], and Frank Leymann[1]

[1] University of Stuttgart
Institute of Architecture of Application Systems
Universitätsstraße 38
70569 Stuttgart
Germany
{mietzner, leymann}@iaas.uni-stuttgart.de
[2] IPL Information Processing Ltd
Eveleigh House, Grove Street
Bath
BA1 5LR
United Kingdom
stefan.pottinger@ipl.com

Abstract. In a Web service world, the Web Services Business Process Execution Language (WS-BPEL) is the standard used to compose Web services into business processes. These processes are often long-running. Therefore WS-BPEL employs a long-running transaction model to handle the internal transactions of a WS-BPEL process. WS-Business Activity (WS-BA) is a set of mechanisms and protocols to coordinate a set of Web Services into a long-running compensation-based transaction. Up to now, it was not possible to let parts of a WS-BPEL process participate in a WS-BA coordination. We show how WS-BA needs to be extended to allow parts of a WS-BPEL process to participate in a WS-BA coordination, which is supervised by an external coordinator. In addition our approach allows external partners to participate in these modified internal WS-BA transactions initiated by a WS-BPEL process and also allows for easy incorporation of BPEL sub-processes into the proposed coordination model. The architecture of a prototype implementing our approach is sketched.

Keywords: WS-BA, BPEL, coordination, long-running transactions, sub-processes.

1 Introduction and Motivation

Web Services enable interactions between heterogeneous business domains based on a common set of specifications. These specifications are part of the Web Service stack [17] and are used to put Web Services into action. The Web Services Business Process Execution Language (WS-BPEL or BPEL for short) [14] is located on top of the Web Service stack. BPEL enables the composition of Web Services into business processes, which are in turn provided as new Web Services. A BPEL process thus orchestrates the Web Services it uses. Besides constructs to define navigation in a process and constructs used for interactions with Web Services, BPEL also provides a long-running transaction model. This transaction model is based on compensation-based recovery. In BPEL4WS [2] this transaction model is described in terms of WS-Business Activity (WS-BA) [15].

WS-BA makes use of the Web Services Coordination (WS-Coordination) Framework [16], which defines basic elements such as a coordinator and a coordination context that can be leveraged by applications built on top of WS-C. From a general point of view, WS-BA defines protocols to support long-running business transactions in a Web Service environment. Thus, WS-BA can be used for any application needing long-running transactions. On the other hand, BPEL, as a specific application domain, uses one specific variant of WS-BA, WS-BA's Business Agreement with Participant Completion coordination protocol, to describe interactions between nested BPEL scopes, BPEL's construct to group activities. But no WS-BA based interaction between a process and the services it uses is assumed.

In this paper we present an approach on how to externalize the coordination between BPEL scopes from a BPEL engine in order to use an external coordinator to manage BPEL's internal transactions. Using an external coordinator has the advantage that external partners of a BPEL process can now participate directly in the transactions of a BPEL process using a standardized coordination framework, namely that of WS-Coordination and the protocols of WS-Business Activity. This has the advantage that no separate compensation activities need to be defined in the process model but the invoked service itself is in charge of undoing its work. Additionally, we extract proprietary implementations of BPEL long-running transactions from BPEL engines and thus provide a more modular approach.

In externalizing the transaction protocol, we show that WS-BA's Participant Completion coordination protocol is only suitable to describe the most important steps in the lifecycle of a BPEL scope but that it cannot be used by a coordinator to coordinate the interactions between nested BPEL scopes without altering BPEL's transaction semantics; the reason is that the coordination protocol lacks states needed for fault and compensation handling. Hence, we extend WS-BA's Participant Completion coordination protocol and show how an external coordinator can manage both, BPEL scopes using the extended coordination protocol and external partners using the standard WS-BA protocol. We will call the extended WS-BA framework WS-Business Activity for BPEL (or WS-BA4BPEL for short). We base our work on the Web Services Business Process Execution Language Version 2.0 standard [14], therefore the term "BPEL" denotes WS-BPEL Version 2.0 in this paper.

Figure 1 shows an example travel booking process. At some point in the process two external partner services, namely the flight booking service and the hotel reservation service, are invoked in parallel in order to book a flight and a hotel.

In case the hotel reservation service fails, we also do not want to book a flight. This scenario can be modeled with normal BPEL. The two invoke activities, book flight and book hotel are placed in one scope (the "Reservation-Scope") and a fault handler is modeled that sends a "cancel hotel booking" message to the hotel reservation service, if the flight booking threw a fault and a “cancel flight reservation” message to the flight booking service if the hotel reservation service threw a fault. However the flight booking service must support a cancel operation and the semantics and syntax of this operation must be known to the process modeler at design time.

Our approach overcomes these limitations by including the external partners in the internal transactions of a BPEL process. A BPEL engine supporting WS-BA4BPEL will start a new long-running transaction when the scope is started that contains the book flight and book hotel invoke activities. Upon invocation of the external partners, the transaction context is submitted to the partners and subsequently the external partners register as participants of the transaction. In case one of the external partners fails, the partner sends a failure message specified in the transaction protocol to the coordinator, which then aborts the other partner using transaction protocol specific messages. Hence, the modeler of the travel booking service does not need to know the syntax or semantic of the cancel operations of the external partner; all that is important is that the external partners conform to WS-BA as they will receive a coordination context and need to know how to register their WS-BA ports with the coordinator.

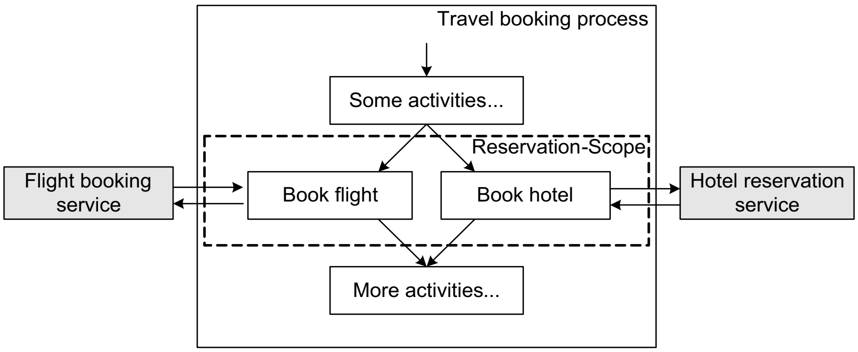


Fig. 1. Example travel booking process

Our paper is organized as follows: We begin with a short overview of BPEL, its transaction model and the basics of WS-Coordination and WS-Business Activity. Afterwards we describe the impacts of using WS-BA including the underlying WS-Coordination framework in order to drive WS-BA4BPEL: The information needed by a WS-BA4BPEL-coordinator to drive BPEL’s transaction model with as little overhead as possible will be explained. We continue by introducing the extensions to WS-BA’s Business Agreement with Participant Completion protocol, so that BPEL’s transaction model for scopes and processes can be managed by a WS-BA4BPEL coordinator. Afterwards we discuss WS-BA4BPEL with respect to sub-processes, transactions not initiated by a BPEL process and partners of a BPEL process included in BPEL’s transaction model. Finally, we present a prototype, related work and topics for future research.

2 BPEL and Its Long-Running Transaction Model

Business processes typically take a long time to complete, e.g. if complex asynchronous interaction patterns are involved or if services are called that take a longer period of time to respond. Therefore ACID transactions can only be applied to work units, which execute fast [7]. ACID transactions are too strong for a business process as it might not be possible to lock resources for an entire business process lasting days. Furthermore, it is not feasible to undo all changes of a business process in case an error occurs shortly before its end [5].
BPEL uses a long-running transaction model based on compensation. This long-running transaction model is defined using the Participant Completion coordination protocol of WS-BA. However, extensions of BPEL may allow ACID transactions [3].

BPEL uses scopes to group activities and to define transactional boundaries. Such scopes are the base of BPEL's transaction model. Besides grouping activities, handlers can be attached to scopes in order to provide actions for particular situations. These handlers are: Event handlers, fault handlers, a termination handler, and a compensation handler. Event handlers react to timing or message events and fault handlers cope with faults occurring inside a scope. A termination handler is called upon forced termination of a scope. If a scope has completed successful, its compensation handler is installed. This compensation handler can be called from the enclosing scope if the enclosing scope is confronted with a fault or is compensating itself. The compensation handler is called using the *compensate* or *compensateScope* activity. A compensate activity specifies the compensation of all nested scopes that have completed where compensateScope activities specify one nested scope that has to be compensated through its `target` attribute. It is important for our approach to emphasize that a compensation handler of a scope can only be called out of a fault or compensation handler of its directly enclosing parent scope. While the behavior of a fault and compensation handler can be explicitly modeled in a BPEL process, BPEL also provides default behavior definitions. In default mode fault and compensation handlers compensate all successfully completed directly nested scopes. After compensation, the default fault handler rethrows the caught fault into the enclosing scope.

BPEL's transaction model defines that whenever a scope is confronted with a fault, all running nested scopes that are not completed and have not seen any fault are terminated. After successful completion, the transaction model defines that a nested scope can be closed, if its enclosing scope is closed as the nested scope can expect not to receive any more messages in the coordination protocol. The BPEL specification [14] lists all possible paths through the life-cycle of scopes. A detailed overview about the BPEL 1.1 transaction model, which did not change in the parts relevant to the discussion in WS-BPEL 2.0, can be found in [4].

2.1 WS-Coordination and WS-Business Activity

WS-Coordination [8][16] provides two main building blocks: *Activities* and *participants*. Activities of WS-Coordination define units of work that participants can take part in. WS-Coordination defines a set of messages that allow applications to create new activities and allow participants to register for these activities.

Once a participant has registered with a WS-C activity it is coordinated by the coordinator using the messages described in a coordination protocol. Upon registration the participant must specify which coordination protocol is used between the participant and the coordinator.

WS-Business Activity [15] specifies two such coordination protocols. Both protocols specify messages needed for the coordination of long-running compensation-based transactions. The coordinator completion protocol is used for participants for which the coordinator decides when they have finished performing their work, while the Participant Completion protocol is used by participants that "know" when they have completed their work.

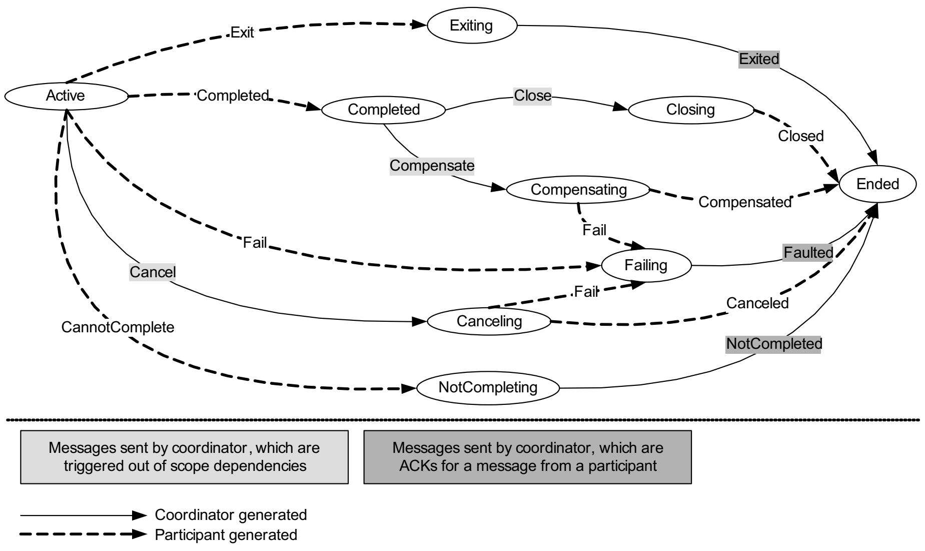


Fig. 2. Business Agreement with Participant Completion Coordination Protocol of WS-BA, states of a participant

Figure 2 shows the states and messages sent and received by a participant using the business agreement with Participant Completion coordination protocol of WS-Business Activity.

3 An External Coordinator to Drive BPEL's Transaction Model

In order to coordinate scope interactions externally, we use a coordinator based on WS-Coordination and an extended WS-Business Activity protocol. WS-Coordination is used to create activities and to register scopes with these activities, thus making them manageable by the external coordinator. Once scopes have been registered as participants of an activity they are coordinated using the messages and states defined in the extended WS-Business Activity as shown in Figure 3. We will show why WS-BA needs to be extended to coordinate BPEL scopes in section 3.2.

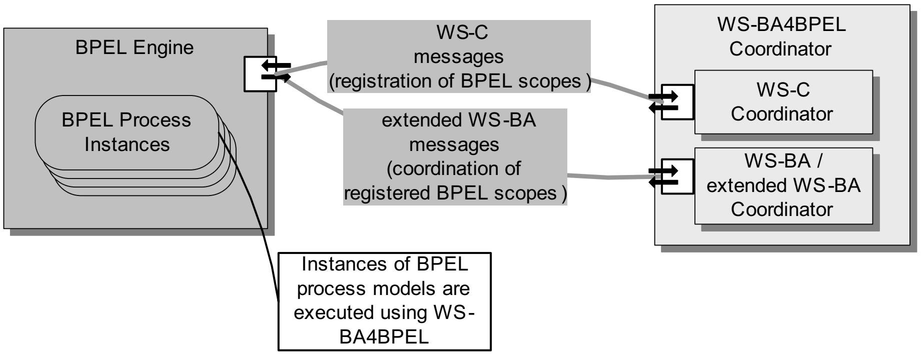

BPEL Engine
BPEL Process Instances
WS-C messages (registration of BPEL scopes)
extended WS-BA messages (coordination of registered BPEL scopes)
WS-BA4BPEL Coordinator
WS-C Coordinator
WS-BA / extended WS-BA Coordinator
Instances of BPEL process models are executed using WS-BA4BPEL

Fig. 3. BPEL-Engine, external coordinator and messages sent

3.1 Mapping of BPEL Scopes to WS-Coordination

An external coordinator coordinating the internal transactions in a BPEL process must be aware of a set of information regarding the process it is coordinating.
First of all, the coordinator must be aware of the parent-child relations of the different scopes. In order to being able to send termination and compensation requests to all nested scopes in case of a fault, the coordinator must be aware which scopes are the children of the scope that just faulted. Additionally, if a fault needs to be rethrown, the coordinator must be aware of the parent of the faulted scope

From the perspective of a WS-Coordination (WS-C) participant, its outcome can depend on the outcome of the WS-C activity it registered with, i.e., WS-C activities might be used to control the outcome of their participants. In a similar way, a BPEL scope takes part in the work of its enclosing scope and can also be controlled by it (that means, the enclosing scope can terminate or compensate it). We therefore propose to create a WS-C activity for every scope that has child scopes. All directly nested scopes of a scope register as participants with that WS-C activity.
Additionally we introduce the concept of a *root WS-C activity* for every process instance. A root WS-C activity is necessary in order to register the BPEL engine as participant representing the *top level scope*, i.e. the process element. This is valid since a process can be implicitly regarded as a scope [14].

WS-Coordination's activity-participant relation is binary. Participants of a WS-C activity representing a scope can therefore be only the scope's directly nested scopes. However, BPEL allows for arbitrary nesting of scopes. In order to reflect this arbitrary scope nesting, we propose the following solution: We use WS-Coordination's `nestedCreate`-mechanism to represent the necessary scope nesting in WS-Coordination. Each scope that is both a nested scope and a scope containing other nested scopes registers as a participant with it's enclosing scope's WS-C activity and also as a nested WS-C activity of it's enclosing scope's WS-C activity. In order to correlate the participant and the activity representing the same scope, we submit a unique identifier for the scope with every creation of an activity and the registration as a participant. A state change of a scope, which has consequences for nested scopes, can therefore be pushed down the hierarchies of scopes to the right participant solely by the coordinator itself. Hence, all coordination logic for BPEL's transaction model can be modularized into the coordinator – thus execution logic and the corresponding transaction model is separated between a BPEL engine and a coordinator driving WS-BA4BPEL. In addition, using the `nestedCreate`-mechanism and unique identifiers connecting activities and participants, the overhead of the coordinated WS-BA4BPEL is reduced, as interactions between a BPEL engine and a coordinator are kept to a minimum.

In case a BPEL long running transaction also includes partners of a BPEL process, which must participate in the transaction of the process, WS-C activities also have to exist for the scopes surrounding the invoking activities even if they are leaf scopes. We discuss the inclusion of partners of a BPEL process in BPEL's transaction model in section 4.

Regarding the flight booking example in section 1, the whole flight-booking process registers with a root activity. Since the process also has child scopes (namely the "reservation-scope") another WS-C activity (the "process-activity") for the process is created. The "reservation-scope" registers as a participant of the "process-activity". Since it contains invocations of external partners, the "reservation-scope-activity" has to be created. When the external partners are invoked, the coordination context of the "reservation-scope-activity" is included in the invocation message. The external partners are expected to register as participants of the "reservation-scope-activity". The endpoint, which indicates where they can register as participants for this transaction, is submitted with the coordination context. Possible child-scopes of the "reservation-scope" also register as participants of the "reservation-scope-activity".

Figure 4 shows the WS-C activity-participant tree for the travel booking example after the external partners have been invoked.

WS-BA offers two different coordination types for an activity: *AtomicOutcome* and *MixedOutcome*. One of those has to be specified upon creation of a new WS-C activity. We propose to create WS-C activities using the mixed outcome coordination type allowing different outcomes among the participants of a WS-C activity. Mixed outcome is used since compensation, fault, and termination handlers containing compensateScope activities can compensate only a subset of the participants of a nested scope, resulting in an activity that has both, completed and compensated participants which corresponds to mixed outcome of WS-C. However there are two exceptions to this rule. The first exception is the root activity. Since this activity only contains one participant which represents the process itself, it is always atomic so it can be created using atomic outcome. The second exception occurs when a scope does not have any defined fault, compensation or termination handlers. If no such handler is contained in the definition of a scope, either all nested scopes complete successfully or all nested scopes are terminated or compensated. This behavior corresponds to atomic outcome of WS-BA. As a result such scopes without defined fault, compensation and termination handlers are created using atomic outcome.

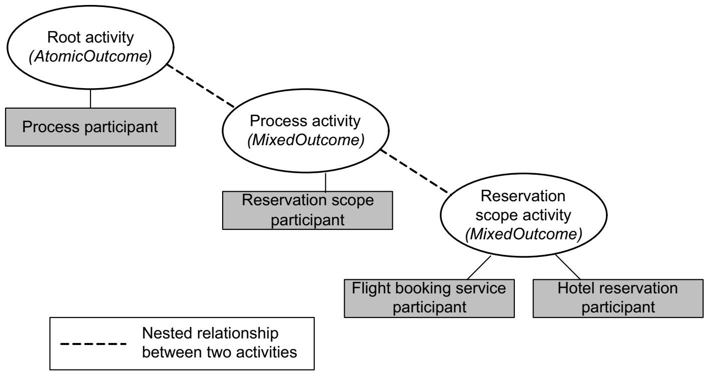

Root activity
(AtomicOutcome)
Process participant
Process activity
(MixedOutcome)
Reservation scope participant
Reservation scope activity
(MixedOutcome)
Flight booking service participant
Hotel reservation participant
Nested relationship between two activities

Fig. 4. WS-Coordination activity-participant tree for the travel booking example

Using atomic or mixed outcome does not make any difference in the behavior of the coordinator in our case, since we will see in section 3.2.2 that compensation in WSBA4BPEL is triggered by participants not by the coordinator. However, using atomic and mixed outcome provides additional information about the behavior of the transaction to external partners that participate in the transaction.

3.2 Modifying WS-BA to Enable Coordinated WS-BA for BPEL

After registration, a scope is coordinated as a participant using our extended Ausiness Agreement with Participant Completion coordination protocol. Participant Completion is used instead of coordinator completion, because a participant representing a BPEL scope knows by itself when it has completed all necessary work. A scope contains all the information to decide whether all of its nested activities have finished and it thus has completed all its work. Upon registration for an activity, the participant indicates that it wants to be coordinated by the extended Business Agreement with Participant Completion protocol by providing a protocol identifier pointing to the extended Business Agreement with Participant Completion protocol. Participants (such as external partners) that need to be coordinated using one of the standard WS-BA protocols can specify those. Therefore we allow participants being coordinated with the extended protocol as well as the standard protocol in the same activity. As illustrated in the original Participant Completion coordination protocol of WS-BA [15] shown in Figure 2, a coordinator can send `Cancel`, `Close` and `Compensate` messages to the participant and respond to `Fail`, `CannotComplete` and `Exit` messages of the participant with the acknowledgements `Failed`, `NotCompleted` and `Exited`.

`Cancel`, `Close` and `Compensate` messages are triggered by events in the lifecycle of an enclosing scope. Therefore the coordinator must have enough information about the state of enclosing scopes to send corresponding messages to the nested scopes.

Regarding the lifecycle of a BPEL scope, the scope itself (or the engine executing the scope instance) must be able to notify the external coordinator of various events in the life-cycle of the scope. Table 1 shows the life-cycle events of a scope that trigger actions in the coordinator and events that are triggered by the coordinator. Additionally, the third column shows the WS-BA messages that correspond to the respective state changes in the lifecycle of a scope.

Table 1. Life-Cycle events of a BPEL scope

Event	**Direction**	**WS-BA Message**
Started scope execution	Scope to coordinator	Handled by WS-C
Scope completed normal processing	Scope to coordinator	`Completed`
Scope failed	Scope to coordinator	Not Available
Terminate scope	Coordinator to scope	`Cancel`
Scope terminated	Scope to coordinator	`Canceled`
Execute fault handler	Coordinator to scope	Not Available
Fault handler completed	Scope to coordinator	`CannotComplete`
Fault handler failed	Scope to coordinator	`Fail`
Compensate scope	Coordinator to scope	`Compensate`
Compensation completed	Scope to coordinator	`Compensated`
Compensation failed	Scope to coordinator	`Fail`
Parent scope has completed	Coordinator to Scope	`Close`

Table 1 shows that not all state changes can be reflected in the Participant Completion coordination protocol of WS-BA. We will show in the following sections how to add the necessary messages and states to this protocol in order to being able to fully control a scope's lifecycle by an external coordinator.

3.2.1 Handling Faults and Canceling Scopes

The original Participant Completion coordination protocol of WS-BA differentiates between three scenarios, which are triggered when a participant is not willing or not able to complete its work successfully. A participant can notify the coordinator through three different messages: `Exit`, `CannotComplete` and `Fail`. An `Exit` message denotes that the participant is no longer willing to participate in the transaction. A `CannotComplete` message denotes that a fault has occurred but that this fault has been handled. In case a scope registered as a participant, a `CannotComplete` message from that participant indicates that the scope has seen a fault but has handled that fault using a fault handler. In the third case, the internal fault handling was not successful and the participant leaves the coordination protocol using the message `Fail`. Which scenario is taken, is not visible for the coordinator until `Exit`, `CannotComplete` or `Fail` is sent. The coordinator has no knowledge about an already occurred fault, until it receives a message from the participant. But a coordinator driving BPEL's transaction model depends on this knowledge as in BPEL active nested scopes must be canceled in the case a fault occurs in their parent scope [14]. Therefore, we introduce the message `CancelSubScopes`, which is directly sent after a fault occurred in the state `Active` and therefore before sending one of the messages `CannotComplete` or `Fail`. Having received `CancelSubScopes`, the coordinator cancels all active nested scopes. The participant is now in state `SubScopesGetCanceled`. After having canceled all active sub-scopes the coordinator responds with a `CanceledSubScopes` message which puts the participant in state `HandlingFault` where it executes its fault handler.

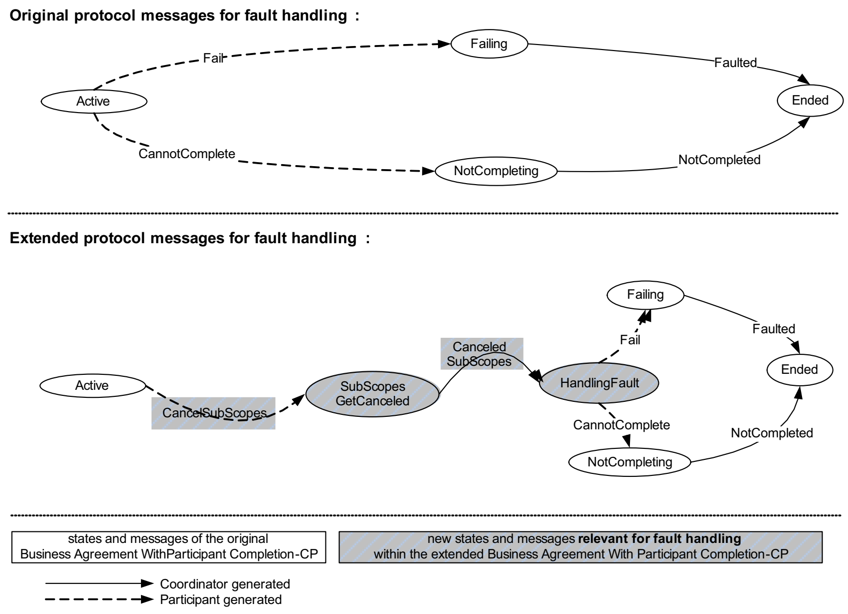

Original protocol messages for fault handling :
Active
Fail
Failing
Faulted
Ended
CannotComplete
NotCompleting
NotCompleted
Extended protocol messages for fault handling :
Active
CancelSubScopes
SubScopes GetCanceled
Canceled SubScopes
HandlingFault
Fail
Failing
Faulted
Ended
CannotComplete
NotCompleting
NotCompleted
states and messages of the original Business Agreement WithParticipant Completion-CP
new states and messages relevant for fault handling within the extended Business Agreement With Participant Completion-CP
Coordinator generated
Participant generated

Fig. 5. Extended WS-BA participant completion coordination protocol: Additional participant states and messages required for fault handling

The described fault handling applies to faults occurring during normal processing of a scope, i.e. scopes in the state Active. This behavior does not apply to faults occurring in fault or compensation handlers. A fault occurring in one of these handlers must be signaled to the coordinator and all active nested scopes of the handler will be aborted by the coordinator.

The message `Fail` is sufficient in this case: It is not possible to associate a fault handler directly to a fault or compensation handler, thus making the `CancelSubScopes` message unnecessary. So we use the message `Fail` out of the states `HandlingFault` and `Compensating` similar to the message `HandlingFault` out of the state `Active` – the abnormal termination of the current processing is signaled.

Figure 5 shows the extensions to WS-BA's Participant Completion coordination protocol with regard to fault handling. States and messages not relevant for fault handling are excluded from Figure 4 for readability.

Regarding the travel booking example introduced in section 1, the message exchange would be the following: During the invocation of the book hotel service an error occurs, the enclosing "reservation scope" sees this error and notifies the coordinator through a `CancelSubScopes` message. We assume that the book flight

Web service is still being executed. The coordinator now sends a Cancel message to the book flight service which eventually responds with Canceled. The coordinator, seeing that all active nested scopes of the "reservation scope" have been canceled, notifies the BPEL engine executing the travel booking process with a CanceledSubScopes message. The engine can now execute the fault handler for the "reservation scope". In our case this is the default fault handler which compensates all nested completed scopes. Since none has completed the engine can now notify the coordinator through a CannotComplete message of the completion of the fault handler. Having received the confirmation from the coordinator the control flow can move on to the activities succeeding the reservation scope.

3.2.2 Compensate

The second type of messages and states introduced by our extended coordination protocol is needed for compensation of nested scopes. Figure 6 shows these messages and states, all other messages and states have been omitted from Figure 6 for readability.

The original WS-BA Participant Completion coordination protocol defines the message Compensate to instruct a participant that it needs to compensate. That participant answers with Compensated or Fail depending whether the compensation has completed successfully or not. There is no message in the original WS-BA Participant Completion protocol allowing a participant to notify the coordinator that other participants should be compensated. In BPEL compensate and compensateScope activities can be used by fault or compensation handlers of a scope to compensate all or one specific nested scope. To reflect this behavior in our extended coordination protocol we introduce the message RequestSubScope-Compensation. This notification message from a participant to the coordinator has to contain the scopes to be compensated. Some of the scopes requested to be compensated may have not finished successfully and thus no compensation handler is installed. Thus the message has the nature of a request and we call the message RequestSubScopeCompensation.

HandlingFault and Compensating are the states of a scope executing a fault or a compensation handler. Scopes nested in a fault or compensation handler may also request compensation of scopes nested in the scope to which the handler belong. The scopes nested in the handlers are in the state Active. Therefore the message RequestSubScopeCompensation may be sent in the states HandlingFault, Compensating, and Active. To reflect the behavior of BPEL's compensate activities, the scope requesting compensation has to be notified by the coordinator that the coordinator has compensated the requested scopes. This leads to RequestSubScopeCompensation messages being answered by Finished SubScopeCompensation messages.

RequestSubScopeCompensation messages requesting an already compensated scope do not have to be considered by the engine, because BPEL 2.0 does not define repeated compensation as an erroneous situation.

Again, we take a look at the travel booking process. Now the scenario is the following. The book flight Web service has completed successfully and has notified

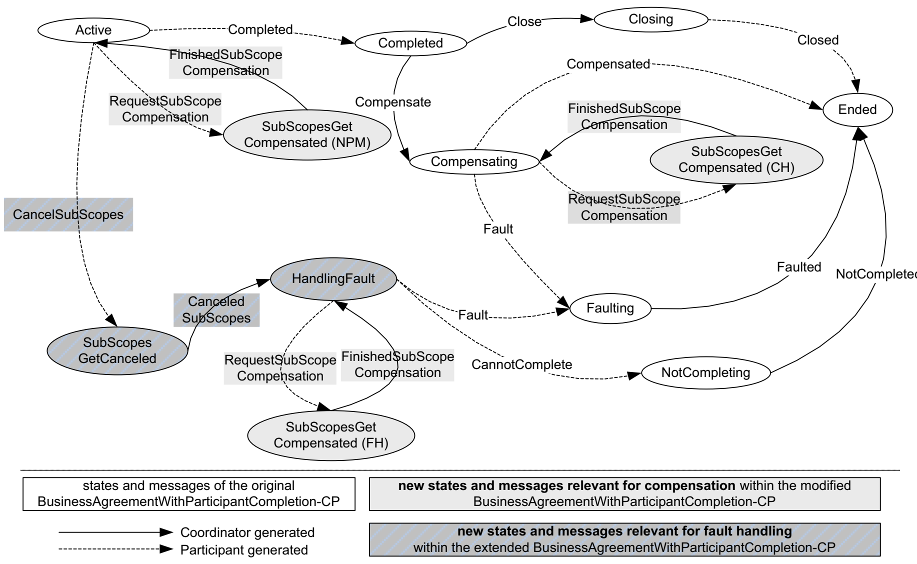

Active
Completed
Completed
Close
Closing
Closed
FinishedSubScope Compensation
Compensated
Ended
RequestSubScope Compensation
SubScopesGet Compensated (NPM)
Compensate
FinishedSubScope Compensation
Compensating
SubScopesGet Compensated (CH)
CancelSubScopes
RequestSubScope Compensation
Fault
Faulted
NotCompleted
HandlingFault
Canceled SubScopes
Fault
Faulting
SubScopes GetCanceled
RequestSubScope Compensation
FinishedSubScope Compensation
CannotComplete
NotCompleting
SubScopesGet Compensated (FH)
states and messages of the original BusinessAgreementWithParticipantCompletion-CP
new states and messages relevant for compensation within the modified BusinessAgreementWithParticipantCompletion-CP
Coordinator generated
Participant generated
new states and messages relevant for fault handling within the extended BusinessAgreementWithParticipantCompletion-CP

Fig. 6. Extended WS-BA participant completion coordination protocol: Additional states and messages required for compensation and fault handling

the coordinator through a `Completed` message of this fact. The book hotel service invocation throws a fault and the fault handler of that service failed, so the coordinator is notified via a fault message. The engine then sees that a fault occurred in the "reservation scope" and sends a `CancelSubScope` message to the coordinator that responds with a `CanceledSubScopes` message (since no active sub-scopes exist at that point). The engine now starts to execute the default fault handler which is specified to compensate all completed nested scopes. The engine must therefore send a `RequestSubScopeCompensation` message to the coordinator which then sends a `Compensate` message to the book flight web-service that responds eventually with a `Compensated` message and now leaves the transaction. The coordinator now tells the reservation scope that all nested scopes have been compensated through a `CompensatedSubScopeMessage`. Then the "reservation-scope" can leave the transaction through an `CannotComplete` message.

3.2.3 Close

A scope is closed if its enclosing scope reaches the state `Ended`. Since compensate activities can only be sent by an enclosing scope, it is certain that the nested scope will not receive a `Compensate` message anymore. The coordinator is informed when the state of a participant changes to `Ended` by a `Closed` message, we need not introduce new messages in order to trigger the sending of `Close`.

As an example, we take a look at the travel booking process. After both, the flight and the hotel booking services have been invoked successfully, the "reservation scope" and the whole have also completed successfully, the coordinator can now notify all the scopes in the process and the external partners that the process has completed and that they can exit from the transaction by sending a `Close` message to the participants of the root scope, i.e., the process participant who then notifies the process activity that it can close, which in turn sends `Close` messages to it's participants and so on until all participants have left the transaction.

4 Discussion and Related Work

So far, we described a modified Participant Completion coordination protocol of WS-BA that can be used to coordinate BPEL's transaction model by an external coordinator. In this section we take a look at our approach in context of partners of a BPEL process, sub-processes [6] and externally initiated WS-BA transactions, discuss interposition of coordinators and present related and future work.

4.1 Including Partners of a BPEL Process in BPEL's Transaction Model

Using a coordinator to drive BPEL's transaction model enables partners of a BPEL process to be incorporated into the process's transactions. They can use the already existing coordinator to register for an activity. Such partners of BPEL processes can choose between the original two coordination protocols of WS-BA, as the changes proposed here only apply to participants that are BPEL scopes. Therefore it makes no difference to a partner of a BPEL process if the process or any other Web Service invites it to join a transaction. A Web Service that is a partner of a BPEL process is not restricted in its behavior if it receives a context for an activity out of a BPEL process compared to an invitation to a WS-BA transaction by a simple Web Service.

In order to provide a partner of a BPEL process with a context, partner links have to be enhanced to define which partners to include in a transaction. This approach has been described in [10] and [12]. While [10] and [12] focus on the inclusion of partners of a BPEL process in transaction models that are not directly connected to BPEL's own transaction model and may be different from it (e.g. WS-Atomic Transaction [13]), our work gives the opportunity to share already existing contexts for BPEL's transaction model with partners of a BPEL process – thus including partners of BPEL processes in BPEL's own transaction model. A different approach to include partners of a BPEL process in a transaction has been made in [11]. The idea is to replace existing compensation handlers with so called coordination handlers that are basically WS-BA coordinators attached to BPEL scopes. These coordination handlers can in turn be used to register partners of processes and participate in BPEL's transaction model. However, replacing every compensation handler with a WS-BA coordinator produces a high overhead compared to the approach discussed in this paper.

4.2 Sub-processes and Externally Initiated WS-BA Transactions

In the preceding sections standalone BPEL process were taken into account. If a BPEL process is executed using WS-BA4BPEL and is also incorporated as a sub-process in another process, this can be also modeled using the approach of WS-BA4BPEL. Instead of the root activity, the activity of the calling scope of the parent process is used. The sub-process then acts as a nested scope of the calling scope. Note that a BPEL engine still has to know when a process is executed as a sub-process so that it can send additional information to the parent process, e.g. “sub-process encountered a terminate activity” as proposed in [6]. If a BPEL process is invited to join a WS-BA transaction, it has to use an already existing context, too. Sub-processes and externally initiated WS-BA transactions share the characteristics that a context is sent to a BPEL process along with the message that kicks off the process. The reused WS-C activity, which is identified by this context, expresses the dependency of the outcome of a BPEL process on the initiator of the BPEL process.

4.3 Implicit Transaction Termination for Root Activity

As just discussed, a BPEL process that depends on the outcome of its initiator is given a context along with its firstly received message. In contrast, an independent BPEL process uses a root activity, which is not nested in another activity, for registration of the top level scope. This knowledge can be used by a coordinator: A standalone BPEL process can be closed successfully by a coordinator in case the process element reaches the state `Completed`. In this case the root activity can not be compensated as no parent activity exists, i.e. a compensation handler of a standalone process that does not take part in an externally initiated transaction can never be invoked. A coordinator driving WS-BA4BPEL can assume that an activity not nested and belonging to a BPEL process, can be closed if the only participant of this activity, i.e. the process, is completed. In this case no explicit termination protocol for the transaction will be necessary. However, in order to make this scenario work, a coordinator needs to be sure that such an activity was created for a BPEL process, which possibly results in new coordination types for WS-BA or in additional information that has to be provided during the creation of a root activity.

4.4 Proof of Concept

We have implemented a prototype based on BPEL 1.1 that demonstrates the approach described above. The open source ActiveBPEL [1] BPEL engine has been modified to support externally coordinated BPEL transactions [9].

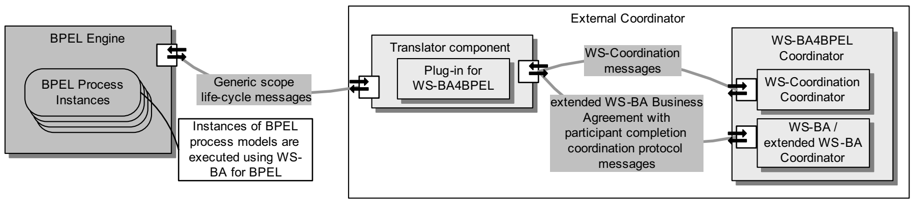

BPEL Engine
BPEL Process Instances
Generic scope life-cycle messages
Instances of BPEL process models are executed using WS-BA for BPEL
External Coordinator
Translator component
Plug-in for WS-BA4BPEL
WS-Coordination messages
extended WS-BA Business Agreement with participant completion coordination protocol messages
WS-BA4BPEL Coordinator
WS-Coordination Coordinator
WS-BA / extended WS-BA Coordinator

Fig. 7. Components and messages involved in the prototype

In order not to limit the prototype to the extended WS-BA coordination protocol, the external coordinator is split into two components, as shown in Figure 7. The first component is a WS-BA4BPEL coordinator that drives the extended Participant Completion coordination protocol. The second component is a translator component that communicates with the BPEL engine through generic messages and translates these messages from and to the coordination protocol messages in order to exchange them with the WS-BA4BPEL coordinator. Both components are implemented as BPEL processes, allowing the communication between the modified BPEL engine and the external coordinator components to be done via SOAP messages, using common bindings, such as a JMS binding in this prototype.

The BPEL engine is modified to send out life cycle messages on the various events during the execution of a BPEL process that need to be communicated to the external coordinator. These events cover the whole life cycle of scopes and their attached handlers. Life cycle messages are sent on the beginning of the execution of a scope, on faulting or completion of a scope as well as execution, faulting and ending of fault and compensation handlers. Having sent out such a life cycle message the modified BPEL engine stops the execution of the scope the event occurred in, and waits for a response from the external coordinator. The modifications in the BPEL engine therefore include the waiting for messages from the external coordinator.

Certain events, namely the rethrowing of a fault into an enclosing scope and the termination and compensation of a scope, are triggered by the external coordinator. They therefore require the ability to trigger events in the life cycle of a scope inside the BPEL engine's navigator from the outside. These events are also supported in the prototype and can be invoked via a Web service interface from the external coordinator. Additionally a monitoring tool has been developed that displays the coordination messages send between the BPEL engine, and the components of the external coordinator.

4.5 Future Work

Our future work involves analyzing the extraction of BPEL's transaction model. In addition, we will revisit WS-BA4BPEL, as e.g. snapshots of variables that are specified for compensation in BPEL 2.0 have not been addressed. While it can be assumed that complex operations, such as snapshots, that heavily depend on a concrete implementation of a BPEL engine, should be implemented in a BPEL engine itself, it will be considered to let a coordinator decide when to take snapshots and where to use them. As a result we would give a coordinator full control over BPEL's transaction model and complete the separation of execution and transaction logic.

In the discussion above, we assumed implicitly that a coordinator processes a `RequestSubScopeCompensation` message, where no scope to be compensated is specified by simply compensating all completed scopes in reverse order of their completion. However, optimizations such as compensating scopes in parallel have not been taken into account [14], since the coordinator is not aware of the control flow between scopes. A possible solution to this problem is to provide a coordinator with a process or scope definition. Nonetheless, as message exchanges between a coordinator and a BPEL engine should be kept to a minimum, we will investigate how much information is needed by a coordinator to fully control WS-BA4BPEL in these cases.

5 Conclusions

BPEL processes can be long-running and provide a transaction model that fits the needs of such long-running business processes. This transaction model is based on compensation and can be described to a certain extend with WS-BA's Business Agreement with Participant Completion coordination protocol.

In this work we demonstrated how to extend WS-BA's Participant Completion protocol so it can be used by a coordinator building on top of WS-Coordination and an extended Participant Completion protocol to manage WS-BA for BPEL. We presented a coordinator that controls the dependencies between scopes, which are introduced by BPEL's transaction model. Thus we clarified the relationship of WS-BA and BPEL and explained the actions a coordinator has to take to coordinate a BPEL process in such a way that the mechanisms of BPEL compensation still apply. In this context we also described the relationship of BPEL scopes and activities of WS-Coordination, discussed when and how WS-C activities have to be created and thus provided an efficient way a coordinator can drive coordinated WS-BA for BPEL without unnecessary overhead.

As our approach builds on modular standards of the Web Service stack, such as WS-Coordination and WS-BA, we showed that it can also be easily integrated with additional scenarios a BPEL process can be found in: Inclusion of external partners, the relationship to sub-processes and the link to WS-BA transactions that are not initiated by a BPEL process are scenarios, which benefit from the approach described in this work.

Acknowledgements

We gratefully acknowledge the valuable input of Oliver Kopp and Dieter Roller for a detailed review of an earlier version of this paper.

References

1. ActiveEndpoints LLC, ActiveBPEL Engine, http://www.activebpel.org/
2. Andrews, T., et al.: Business Process Execution Language for Web Services Version 1.1 (2003), http://www.ibm.com/developerworks/library/ws-bpel/
3. Blow, M., Goland, Y., Kloppmann, M., Leymann, F., Pfau, G., Roller, D., Rowley, M.: BPELJ: BPEL for Java Technology, BEA Systems and IBM Corporation (2004), http://www.ibm.com/developerworks/library/specification/ws-bpelj/
4. Curbera, F., Khalaf, R., Leymann, F., Weerawarana, S.: Exception Handling in the BPEL4WS Language. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 276–290. Springer, Heidelberg (2003)
5. Gray, J.: The transaction concept: Virtues and limitations (invited paper). In: Proceedings of the VLDB, pp. 144–154. IEEE Computer Society, Los Alamitos (1981)
6. Kloppmann, M., Koenig, D., Leymann, F., Pfau, G., Rickayzen, A., von Riegen, C., Schmidt, P., Trickovic, I.: WS-BPEL Extension for Subprocesses (BPEL-SPE), IBM Corporation and SAP AG (2005)

7. Leymann, F., Roller, D.: Production Workflow. Prentice Hall, Upper Saddle River, New Jersey (2000)
8. Leymann, F., Pottinger, S.: Rethinking the Coordination Models of WS-Coordination and WS-CF. In: IEEE ECOWS 2005. Proceedings of the 3rd IEEE European Conference on Web Services, Vaxjö, Sweden (2005)
9. Mietzner R.: Extraction of WS-BA from BPEL 1.1, University of Stuttgart, Diploma Thesis (2006), http://elib.uni-stuttgart.de/opus/volltexte/2006/2864/
10. Mikalsen, T., Khalaf, R., Tai, S.: Composition of coordinated Web Services. In: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware (2005)
11. Sauter, P., Melzer, I.: A Comparison of WS-BusinessActivity and BPEL4WS Long-Running Transaction. In: KIVS 2005, Kaiserslautern, Germany (2005)
12. Tai, S.: Composing Web Services Specifications: Experiences in Implementing Policy-driven Transactional Processes. In: Proceedings of GI BTW 2005, Karlsruhe, Germany (2005)
13. Organization for the Advancement of Structured Information Standards (OASIS), Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.1 (2007)
14. Organization for the Advancement of Structured Information Standards (OASIS), Web Services Business Process Execution Language Version 2.0 (2007)
15. Organization for the Advancement of Structured Information Standards (OASIS), Web Services Business Activity (WS-Business Activity) Version 1.1 (2007)
16. Organization for the Advancement of Structured Information Standards (OASIS), Web Services Coordination (WS-Coordination) Version 1.1 (2007)
17. W3C Working Group Note, Web Services Architecture (2004), http://www.w3.org/TR/ws-arch/

All links last followed on 2007-06-26.

Verifying Composite Service Transactional Behavior Using Event Calculus*

Walid Gaaloul[1], Mohsen Rouached[2], Claude Godart[2], and Manfred Hauswirth[1]

[1] DERI-NUIG
IDA Business Park, Galway, Ireland
{walid.gaaloul,manfred.hauswirth}@deri.org
[2] LORIA-INRIA-UMR 7503
BP 239, F-54506 Vandœuvre-les-Nancy Cedex, France
{rouached,godart}@loria.fr

Abstract. A key challenge of Web service (WS) composition is how to ensure reliable execution. The lack of techniques that support non-functional features such as execution reliability is widely recognized as a barrier preventing widespread adoption. Therefore, there is a growing interest for verification techniques which help to prevent WS composition execution failures.

In this paper, we propose an event driven approach to validate the transactional behavior of WS compositions. Using the Event Calculus to formally specify and check the transactional behavior consistency of WS composition, our approach provides a logical foundation to ensure recovery mechanisms consistency at design time and report execution deviations after runtime.

1 Introduction

Service oriented architecture is gaining prominence as a key architecture to support BPM (Business Process Management) and integrate applications in diverse and heterogeneous distributed environments. Enterprises are able to outsource their internal business processes as services and make them accessible via the Web. Then, they can dynamically combine individual services to provide new value-added WS compositions or composite services (CS). It is widely recognized that one of the barriers preventing widespread adoption of this technology is a lack of products that support non-functional features of applications, such as execution reliability. Due to the inherent autonomy and heterogeneity of Web services, the guarantee of correct CS executions remains a fundamental problem issue. Service execution reliability is a challenging aspect of service composition that has not been deeply investigated so far despite its importance.

In order to ensure a correct and reliable CS execution, our interest is on analysing and checking Web service transactional behavior consistency. An execution is correct

* The work presented in this paper was supported by the EU funding under the SUPER project (FP6-026850) and by the Lion project supported by Science Foundation Ireland under Grant No. SFI/02/CE1/I131.

if it reaches its objectives or fails (properly) according to the designers requirements. In this paper, we propose a formalism based on the Event Calculus (EC) [1] for specifying CS failure handling policies and formally validate the transactional behavior of the CS model, and for checking and analyzing CS execution consistency. EC is interesting because it supports the direct representation of events that are used in such policies, and the advantage of such a formalism is that it facilitates a common representation for transactional behavior and supports the same logical foundation for verification at both design time and after execution time.

The transactional behavior verification can be done either a-priori, i.e., at design time, or a-posteriori, i.e., after runtime to test and repair design errors, and formally verify whether the process design does have certain desired properties. For the a-priori part of the work, we must be able to express CS using a formalism we can reason on. For the a-posteriori part of the verification, CS execution should be auditable by providing functionalities to collect execution logs. In our approach, WS logs are used not for mining but for a-posteriori verification of CS transactional behavior.

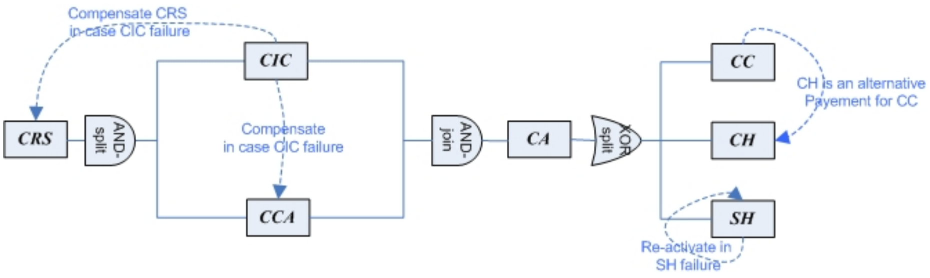

Compensate CRS in case CIC failure
CIC
CRS
AND-split
Compensate in case CIC failure
CCA
AND-join
CA
XOR split
CC
CH
SH
CH is an alternative Payement for CC
Re-activate in SH failure

Fig. 1. CRB illustrative example

In the following, we introduce a scenario to illustrate our approach. Let us consider a Car Rental Broker (CRB) application (Figure 1). This CS acts as a broker offering its customers a set of choices during the Customer Requirements Specification (CRS) service. The CIC service checks the customer ID while the Car Checking Availability (CCA) service provides available cars information and the respective car rental companies. Afterwards, the customer makes his choice and agrees on rental terms in the CA service. Finally, the customer is requested to pay either by credit card (CC), by check (CH), or cash (SH). To deal with failures, the designers of the composite service may augment this control flow with a set of transactional requirements. For instance, they may require to compensate CCA and CRS if CIC fails. Also, they may specify CH as an alternative if CC fails and SH should be re-executed until success in case of failure. Failure handling mechanisms are not provided for the other services that are supposed to never fail. The main problem is how to ensure that the specified CS model is consistent or valid and guaranties reliable executions.

In this paper, we introduce in Section 2 our transactional CS model. In Section 3, we present how we specify the transactional behavior using the EC. The validation of our model is explained in section 4. Section 5 is dedicated to implementation issues. Finally, section 6 outlines some related works before concluding.

2 Transactional Behavior in Composite Web Services

In the loosely coupled environment represented by Web services, long running applications will require support for recovery and compensation, because machines may fail, processes may be cancelled, or services may be moved or withdrawn. Web services transactions also must span multiple transaction models and protocols native to the underlying technologies onto which the Web services are mapped. However, handling failures using the traditional transactional model for long running, asynchronous, and decentralized activities has been proven to be unsuitable. Advanced Transaction Models (ATMs) [2] have been proposed to manage failures, but, although powerful and providing a nice theoretical framework, ATMs are too database-centric, limiting their possibilities and scope [3] in this context (e.g. their inflexibility to incorporate different transactional semantics as well as different behavioural patterns into the same structured transaction). In the same time, workflow has became gradually a key technology for business process automation [3], providing a great support for organizational aspects, user interface, monitoring, accounting, simulation, distribution, and heterogeneity. In our transactional CS model, we propose to combine workflow flexibility and transactional reliability to specify and orchestrate reliable Web services compositions.

In this section, we show how we combine a set of transactional services to formally specify the transactional CS model in Event calculus. We illustrate in particular how we model the composed services scheduling at various levels of abstraction. At the beginning, we show how a transactional CS defines axioms on the transitions of its composing services (Section 2). Then, we show how these axioms enable to express services dependencies on a higher level of abstraction (Section 2.3). These dependencies define the control flow (workflow flexibility) and the transactional flow (transactional reliability).

2.1 Event Calculus

Our approach uses the Discrete EC proposed by Mueller [4] as an extension of the classic EC, to declaratively model event based requirements specifications. The approach, that we present, adopts an EC reasoning to verify CS transactional behavior. This approach is defined by the specification of the axioms describing the transitions carried out and their effects on the services states during CS instance executions. Compared to other formalisms, the choice of EC is motivated by both practical and formal needs, and gives several advantages. From a formal point of view we have three major advantages: First, in contrast to pure state-transition representations, the EC ontology includes an explicit time structure. This helps managing event-based systems where a number of input events may occur simultaneously. Second, the EC ontology is close enough to popular standards like WSBPEL support automatic into the logical representation. Thus, we use the same logical foundation for verification at design time (a-priori analysis) and after runtime (a-posteriori analysis). Third, the semantics of non-functional requirements can be represented in EC, so that verification is once again straightforward.

We adapt a simple classical logic form of the EC, whose ontology consists of (i) a set of time-points isomorphic to non-negative integers, (ii) a set of time-varying properties called fluents, and (iii) a set of event types. The logic is correspondingly sorted, and includes the predicates $Happens$, $Initiates$, $Terminates$ and $HoldsAt$, as well as some auxiliary composite predicates defined in terms of these. $Happens(a,t)$ indicates that event a actually occurs at time-point t. $Initiates(a,f,t)$ (resp. $Terminates$ (a,f,t)) means that if event a were to occur at t it would cause fluent f to be $true$ (resp. $false$) immediately afterwards. $HoldsAt(f,t)$ indicates that fluent f is true at t.

2.2 Transactional Web Service Model

Definition 1 (Transactional Service). *A transactional service, t_s, is a triplet t_s = ($ID \in \mathcal{O}bject$, $E \subset \mathcal{S}tates$, $T \subset \mathcal{T}ransition$) where ID is an object designating service ID, E is the set of its states and T is the set of transitions performing the changes between states. We denote $\mathcal{TWS}$ as the set of all transactional Web services.*

By Web service we mean a self-contained modular program that can be discovered and invoked across the Internet. Each service (Definition 1) can be associated to a life cycle statechart. A set of of states (*initial, active, cancelled, failed, compensated, completed*) and a set of transitions (*activate(), cancel(), fail(), compensate(), complete()*) are used to describe the service status and the service behavior. A transition[1] is performed through two predicates: The first predicate initializes a new state e_2 ($initiates$ (t_s.*tr()*,$e_2(t_s)$, t1)) and the second predicate finishes another state e_1 ($terminates$ (t_s.*tr()*, $e_1(t_s)$, t2)); t2<t1. For instance, the transition *cancel()* initializes a new state *cancelled* by the predicate ($initiates$ (t_s.*cancel()*,$cancelled(t_s)$, t1)) and finishes another state by the predicate *active* by the predicate ($terminates$ (t_s.*cancel()*, $active(t_s)$, t2)); t2<t1.

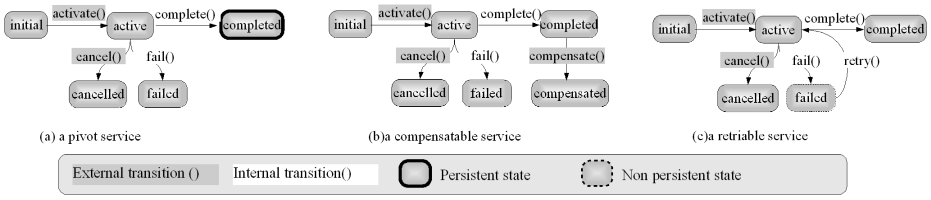


Fig. 2. State transitions diagrams of transactional Web services properties

We distinguish between internal (intra-service) transitions (*complete(), fail(), and retry()*) and external (inter-services) transitions (*activate(), cancel(), and compensate()*). External transitions are fired by external entities (other services, human actor, etc.). Typically they allow a service to interact with the outside and to specify composite services orchestration. The internal transitions are fired by the service itself (the service agent) and are invariably defined. The internal service behavior is refined to express the service transactional properties. The main transactional properties [5] of a Web service we are considering are *retriable, compensatable and pivot* (see Figure 2). A service ts

[1] t_s.*tr()* denotes the execution of the transition *tr()* on the service ts and $e(ts)$ indicates the state e of t_s.

is said to be ***retriable*** (ts^r) if it is sure to complete after finite activations. ts is said to be ***compensatable*** (ts^c) if it offers compensation policies to semantically undo its effects. ts is said to be ***pivot***(ts^p) if once it successfully completes, its effects remain and cannot be semantically undone. Naturally, a service can combine properties, and the set of all possible combinations is $\{r; cp; p; (r, cp); (r, p)\}$.

Table 1 specifies the predicates of the transactional service properties. ts^p's predicates show that the *completed* state is a persistent state. ts^c's predicates describe a transition from the *completed* to *compensated* states. This transition is performed as a part of an other service recovery process. ts^r's predicates ensure that the *retry()* transition brings the service to *active* state each time it fails. Consequently the *failed* state is not a persist state and cannot be a final state for the service.

Table 1. Transactional service properties predicates

Properties	**Predicates**
ts^p	$\nexists(a() \in \mathcal{T}ransitions(ts^p)) \mid (terminates(ts^p$.a() ,*complete*($ts^p$), t))$\leftarrow$ $HoldsAt$(*complete*(ts^p),t) $\wedge$ $Happens(ts^p.a()$,t)
ts^c	$\exists(sf \in \mathcal{TWS} \wedge sf \neq ts^c \mid (Happens(ts^c$.*compensate()*, t2) $\leftarrow$ $HoldsAt$(*fail*(sf), t1))) $\wedge t1 < t2 \wedge$ ($initiates(ts^c$.*compensate()*, *compensated*(ts), t1) $\wedge$ $terminates(ts^c$.*compensate()()*, *completed*(ts), t1) $\leftarrow$ $HoldsAt$(*completed*(ts^c),t)$\wedge$ $Happens$(ts.*compensate()*, t)) $\wedge$t<t1
ts^r	$Happens(ts^r$.*retry()*, t1) $\leftarrow$ $HoldsAt$(*failed*(ts^r), t)$\wedge$t<t1

2.3 Transactional Composite Service Model

A composite service is a conglomeration of existing Web services working in tandem to offer a new value-added service [6]. It orchestrates a set of services, as a composite service to achieve a common goal. A transactional composite (Web) service (TCS) is a composite service composed of transactional services. Such a service takes advantage of the transactional properties of component services to specify failure handling and recovery mechanisms. Concretely, a TCS implies several transactional services and describes the order of their invocation, and the conditions under which these services are invoked.

Definition 2 (Transactional Composite Service: TCS). *A Transactional Composite service t_{cs} is tuple $t_{cs} = (WA \subset \mathcal{TWS}, Ax \subset \mathcal{P}redicate)$ where WA is the set of its component transactional services and Ax is the set of predicates defined by the function $\mathcal{A}xm$ which defines for each inter-services transition* tr() *of a service s, the related invoking predicate:*

$\mathcal{A}xm$: $\mathcal{T}rs_{Itr} \longrightarrow \mathcal{P}redicate_{t_{cs}}$

a.tr() $\longmapsto \mathcal{A}xm(a$.tr()) = (*happens*($a$.tr(),$t$) $\leftarrow \wedge_{i=1..n} P_i$); P_i *is an EC predicate.*

where $\mathcal{T}rs_{Itr}$ is the set of inter-services transitions of t_{cs}'s composing services and $\mathcal{P}redicate_{Inter}$ is the set of predicates relating only on transitions produced by t_{cs} composing services. We define the function $Pdc_t(a$.tr()) that returns the P_i set.

A TCS defines a set of predicates on each component service's external transition in order to define the orchestration. These predicates specify for each component service when it will be activated, cancelled, or compensated. More formally we define a TCS as the set of its composing services and the set of predicates defined on their external transition (see Definition 2). The function $\mathcal{A}xm$ defines for each component service's external transition $t()$ the predicates that induce the event reporting the enactment of this transition. For example, the CRB service specifies that CA will be activated after the completion of CIC and CCA. That means that $\mathcal{A}xm((CA.activate())$ = $happens(CA.activate(), t3) \leftarrow happens(CIC.complete()\ t2) \wedge happens(CCA.complete(), t1) \wedge t1 < t3 \wedge t2 < t3$.

Predicates express at a higher abstract level relations between component services in form of dependencies (*cf.* Definition 3). These dependencies express how services are coupled and how the behavior of certain component service(s) influences the behavior of other one(s).

Definition 3 (Services dependency). *Let cs be a TCS, s_1 and s_2 two component services of cs, $s_1.t_1()$ an transition of s_1, and $s_2.t_2()$ an* ***external*** *transition of s_2, a dependency from $s_1.t_1()$ to $s_2.t_2()$, denoted $dep(s_1.t_1(), s_2.t_2())$, exists if the activation of $s_1.t_1()$ may fire the activation of $s_2.t_2()$. More formally:*

$$\exists dep(s_1.t_1, s_2.t_2()) \Leftrightarrow happens(s_1.t_1(), t1) \in Pdc_t(s_2.t_2()) \wedge t < t1$$
$$\overset{\text{Definition2}}{<===>} \mathcal{A}xm(s_2.t_2()) = (happens(s_2.t_2(), t) \leftarrow happens(s_1.t_1(), t1) \wedge (\wedge_{i=1..n-1} P_i)) \wedge t < t1$$

We distinguish between "normal" execution dependencies and "exceptional" or "transactional" execution dependencies which express the control flow and the transactional flow respectively. The control flow defines a partial services activations order within a composite service instance where all services are executed without failing, cancelled or suspended. Formally, we define a control flow as TCS whose dependencies are only "normal" execution dependencies (*cf.* Definition 4). For example, CRB service defines a "normal" activation dependency from CIC and CCA, to CA such that CA will be activated after the completion of CIC and CCA. That means there are two normal dependencies: $depNrm(CIC, CA)$ and $depNrm(CCA, CA)$

Definition 4 (Control flow). *A* control flow *of a TCS t_{cs} = $(WA, \mathcal{A}xm)$ is the 2-tuple $fc_{t_{cs}}$ = $(AFC, \mathcal{A}xm_{cf})$ inheriting from Definition 2. $AFC \subseteq WA$ and $\mathcal{A}xm_{cf} \subseteq \mathcal{A}xm$ are respectively the services set and related predicates that define only normal execution dependencies between t_{cs} services. A normal execution dependency, denoted $depNrm(s_1, s_2)$, from s_1 to s_2 exists* iff *the completion of s_1 may fire the activation of s_2. More formally :*

$$depNrm(s_1, s_2) \overset{\text{def}}{=} dep(s_1.\text{complete}(), s_2.\text{activate}()).$$

The transactional flow describes the transactional dependencies which specify the recovery mechanisms applied following services failures (*i.e.* after *fail()* transition). We distinguish between different transactional dependencies types (compensation, cancelation and alternative dependencies) (*cf.* Definition 5). Alternative dependencies allow to define a forward recovery mechanisms. A compensation dependency allows to define a backward recovery mechanism by compensation. A cancelation dependency allows to signal a service execution failure to other service(s) being executed in parallel by canceling their execution. For instance the CRB composite service shown in Figure 1 defines an alternative dependency from CC to CH such that CH will be activated when CC fails. That formally means $depAlt(CC, CH) \stackrel{\text{def}}{=} dep(CC.fail(), CH.activate()) \Leftrightarrow \exists happens(CC.fail(), t1) \in Pdc_t(CH.activate()) \wedge t < t1$.

Definition 5 (Transactional Flow). *A* transactional flow *of a TCS* $t_{cs} = (WA, \mathcal{A}xm)$ *is the 2-tuple* $ft_{t_{cs}} = (AFC, \mathcal{A}xm_{tf})$ *inheriting from Definition 2.* $AFC \subseteq WA$ *and* $\mathcal{A}xm_{tf} \subseteq \mathcal{A}xm$ *are respectively the services set and related predicates that define only transactional execution dependencies between* t_{cs} *services which is subdivided are four types:*

1. *Alternative dependency:* $depAlt(s_1, s_2) \stackrel{\text{def}}{=} dep(s_1.\text{fail}(), s_2.\text{activate}())$
2. *Compensation dependency* $depCps(s_1, s_2) \stackrel{\text{def}}{=} dep(s_1.\text{fail}(), s_2.\text{compensate}()) \vee dep(s_1.\text{compensate}(), s_2.\text{compensate}())$.
3. *Cancelation dependency* $depCnl(s_1, s_2) \stackrel{\text{def}}{=} dep(s_1.\text{fail}(), s_2.\text{cancel}())$.

3 Transactional WS Patterns

The use of workflow patterns [7] appears to be an interesting idea to compose Web services. However, current workflow patterns do not take into account the transactional properties (except the very simple cancellation patterns category [8]). It is now well established that the transactional management is needed for both composition and coordination of Web services. That is the reason why the original workflow patterns were augmented with transactional dependencies, in order to provide a reliable composition [9].

In this section, we use workflow patterns to describe TCS's control flow model as a pattern composition. Afterwards, we extend them in order to specify TCS's transactional flow, in addition to the control flow they are considering by default. Indeed, the transactional flow is tightly related to the control flow. The recovery mechanisms (defined by the transactional flow) depends on the execution process logic (defined by the control flow). For example, regarding the CRB composite service, it is possible to define CH as an alternative to CC because (according to the XOR-split control flow operator) they are defined on exclusive branches. Similarly, it is possible to define a compensation dependency from CIC to CCA because (according to the AND-join control flow operator) the failure of CIC requires the compensation of the work already done in the executed service CCA.

Thus, the transactional flow should respect some consistency rules given a control flow. In the following we formally specify these patterns and related transactional consistency rules using EC. These rules are inspired from [9] which specify and prove the potential transactional dependencies of workflow patterns. Due to the lack of space, we put emphasis on the following three patterns *AND-split*, *AND-join* and *XOR-split* (see Figure 3) to explain and illustrate our approach, but the concepts presented here can be applied to other patterns[2].

[2] Our approach also considers the following list of patterns: sequence, ANDsplit, OR-split, XOR-split, AND-join, OR-join, XOR-join and m-out-of-n.

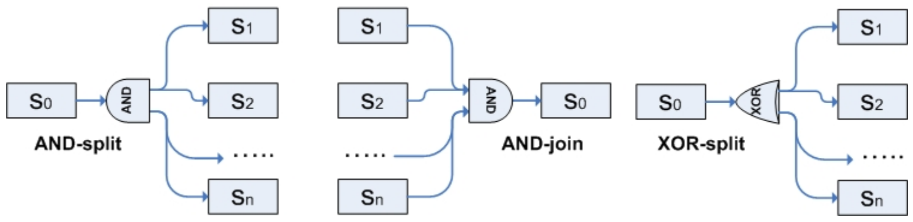


Fig. 3. Studied patterns

An AND-split pattern defines a point in the process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing services to be executed simultaneously or in any order (Definition 7). Our motivating example illustrates the control flow result of the application of AND-split pattern to the set of services $\{CRS, CIC, CCA\}$. The consistency transactional rules (see Table 2) of the AND-split pattern support only compensation dependencies from s_i $(1 \leq i \leq n)$ (rule 1), as the compensation dependencies can be applied only over already activated services. The consistency transactional rules support also cancellation dependencies (rule 2) between only the concurrent services s_i $(1 \leq i \leq n)$, as the cancellation dependencies can exist only between parallel services. Any other cancellation or alternative or compensation dependencies between the pattern's services (rule 3, 4) are forbidden.

Definition 6 **(*AND-split* pattern).** *We define the AND-split pattern as the function:*
AND-split: $\mathcal{P}(\mathcal{WTS}) \longrightarrow fc_{t_{cs}}$
$\{s_0, s_1, s_2, \ldots, s_n\} \longmapsto fc_{ANDsplit} = (S, \mathcal{A}xm_{ANDsplit})$ *such as*
- $S = \{s_0, s_1, s_2, \ldots, s_n\}$,
- $\forall i, 1 \leq i \leq n\ depNrm(s_0, s_i)$
- *Predicates of* s_0: $\mathcal{A}xm(s_0.\text{activate}())$ = predicates outside $fc_{ANDsplit}$
- *Predicates of* $\{s_1, s_2, \ldots, s_n\}$: $\forall i,\ 1 \leq i \leq n\ \mathcal{A}xm(s_i.\text{activate}())$ = $happens(s_i.\text{activate}(), t2) \leftarrow happens(s_0.\text{complete}(), t1) \wedge t1 < t2$.

Definition 7 **(*AND-join* pattern).** *We define the AND-join pattern as the function:*
AND-join: $\mathcal{P}(\mathcal{WTS}) \longrightarrow fc_{t_{cs}}$
$\{s_1, s_2, \ldots, s_n, s_0\} \longmapsto fc_{ANDjoin} = (S, \mathcal{A}xiome_{ANDjoin})$ *such that*
- $S = \{s_1, s_2, \ldots, s_n, s_0\}$,
- $\forall i, 1 \leq i \leq n\ depNrm(s_i, s_0)$
- *Predicates of* s_0: $\mathcal{A}xm(s_0.\text{activate}())$ = $happens(s_0.\text{activate}(), t2) \leftarrow \bigwedge_{i=1\ldots n} happens(s_i.\text{complete}(), t1) \wedge t1 < t2$
- *Predicates of* $\{s_1, s_2, \ldots, s_n\}$: $\forall i,\ 1 \leq i \leq n\ \mathcal{A}xm(s_i.\text{activate}())$ = predicates outside $fc_{ANDjoin}$.

An AND-join pattern defines a point in the process where multiple parallel subprocesses/services converge into one single thread of control, thus synchronizing multiple

Table 2. Transactional consistency rules of patterns

Patterns	**Transactional consistency rules**
AND-split	1. $(\forall s_i \in S \mid 0 \leq i \leq n \ Pdc_t((s_i.\mathit{compensate()})) \subset \{ \ happens(s_j.\mathit{compensate()},\text{t1}), happens(s_j.\mathit{fail()},\text{t2}) \mid 1 \leq j \leq n; i \neq j \ \} \wedge t1 < t, t2 < t;)$
	2. $(\forall s_i \in S \mid 1 \leq i \leq n \ Pdc_t((s_i.\mathit{cancel()})) \subset \{ \ happens(s_j.\mathit{fail()},\text{t1}) \mid 1 \leq j \leq n \, ; i \neq j \ \} \wedge t1 < t;)$
	3. $(\nexists s \in S \mid happens(s.\mathit{fail()},\text{t1}) \in Pdc_t((s_i.\mathit{activate()})) \wedge t1 < t; 0 \leq i \leq n)$
	4. $(\nexists s \in S \mid happens(s.\mathit{fail()},\text{t1}) \in Pdc_t((s_0.\mathit{cancel()})) \wedge t1 < t;)$
AND-join	1. $(\forall s_i \in S \mid 1 \leq i \leq n \ Pdc_t((s_i.\mathit{compensate()})) \subset \{ \ happens(s_j.\mathit{compensate()},\text{t1}), happens(s_j.\mathit{fail()},\text{t2}) \mid 0 \leq j \leq n; i \neq j \ \} \wedge t1 < t, t2 < t;)$
	2. $(\forall s_i \in S \mid 1 \leq i \leq n \ Pdc_t((s_i.\mathit{cancel()})) \subset \{ \ happens(s_j.\mathit{fail()},\text{t1}) \mid 1 \leq j \leq n \ \} \wedge t1 < t;)$
	3. $(\nexists s \in S \mid happens(s.\mathit{fail()},\text{t1}) \in Pdc_t((s_i.\mathit{activate()})) \wedge t1 < t \ 0 \leq i \leq n)$
	4. $\nexists s \in S \mid happens(s.\mathit{compensate()},\text{t1}) \vee happens(s.\mathit{fail()},\text{t1}) \in Pdc_t((s_0.\mathit{compensate()})), t1 < t$
	5. $\nexists s \in S \mid happens(s.\mathit{fail()},\text{t1}) \in Pdc_t((s_0.\mathit{cancel()})), \wedge t1 < t$
XOR-split	1. $\forall s_i \in S \mid 1 \leq i \leq n \ Pdc_t((s_i.\mathit{activate()})) \subset \{ \ happens(s_0.\mathit{completed()},\text{t}), happens(s_j.\mathit{fail()},\text{t}) \mid 1 \leq j \neq i \leq n \ \}$
	2. $(Pdc_t(s_0.\mathit{compensate()}) \subset \{ \ happens(s_j.\mathit{compensate()},\text{t1}), happens(s_j.\mathit{fail()},\text{t2}) \mid 0 \leq j \leq n \ \} \wedge t1 < t, t2 < t;)$
	3. $(\nexists s \in S \mid happens(s.\mathit{fail()},\text{t1}) \in Pdc_t((s_i.\mathit{cancel()})) \wedge t1 < t\text{:} \ 0 \leq i \leq n)$
	4. $(\nexists s \in S \mid happens(s.\mathit{fail()},\text{t1}) \vee happens(s.\mathit{compensate()},\text{t1}) \in Pdc_t((s_i.\mathit{compensate()})) \wedge t1 < t \wedge 1 \leq i \leq n)$
	5. $(\nexists s \in S \mid happens(s.\mathit{fail()},\text{t1}) \in Pdc_t((s_0.\mathit{activate()})) \wedge t1 < t)$

threads (Definition 8). Our example illustrates the control flow result of the application of AND-join pattern to the set of services $\{CIC, CCA, CA\}$. The consistency transactional rules (see table 2) of the AND-join pattern support only compensation dependencies for s_i $(1 \leq i \leq n)$ (rule 1). Indeed, s_0 can not be compensated by s_i $(1 \leq i \leq n)$ as they are executed after. The consistency transactional rules support also cancellation dependencies (rule 2) between only the concurrent services s_i $(1 \leq i \leq n)$, as the cancellation dependencies can exist only between concurrent services. Any other cancellation or alternative or compensation dependencies between the pattern's services (rule 3, 4, 5) are forbidden.

Definition 8 **(*XOR-split* pattern).** *We define the XOR-split pattern as the function:*

XOR-split*:* $\mathcal{P}(\mathcal{WTS}) \longrightarrow fc_{t_{cs}}$

$\{s_0, s_1, s_2, \ldots, s_n\} \longmapsto fc_{XORsplit} = (S, \mathcal{A}xiome_{XORsplit})$ *such that*

- $S = \{s_0, s_1, s_2, \ldots, s_n\}$,
- $\forall i, 1 \leq i \leq n \ depNrm(s_0, s_i)$
- *Predicates of* s_0*:* $\mathcal{A}xm(A.\text{activate()})$ = predicates outside $fc_{XORsplit}$
- *Predicates of* $\{s_1, s_2, \ldots, s_n\}$*:* $\forall i, \ 1 \leq i \leq n \ \mathcal{A}xm(s_i.\text{activate()}) =$ *happens*$(s_i.\text{activate()},t2) \leftarrow$ *happens*$(s_0.\text{complete()}, t1) \wedge t1 < t2 \wedge$ *happens*$(c_i,t) \mid$ *there is always only one condition* c_j $(1 \leq j \leq n)$ *evaluated to true after* s_0 *termination.*

An XOR-split pattern defines a point in the process where, based on a decision or control data, one of several branches is chosen (Definition 8). Our example illustrates the control flow result of the application of XOR-split pattern to the set of services $\{CA, CC, CH, SH\}$. The consistency transactional rules (see table 2) of the XOR-split pattern support alternative dependencies (rule 1) between only the services s_i $(1 \leq i \leq n)$, as the alternative dependencies can exist only between parallel and non concurrent flows. The consistency transactional rules support also compensation dependencies (rule 2) from (s_i, $1 \leq i \leq n$) to s_0. Any other cancellation or alternative or compensation dependencies between the pattern's services (rule 3, 4, 5) are forbidden.

4 Event-Based Transactional Behavior Validation

In the previous section, we showed how to formally specify a TCS using Event Calculus predicates. The objective of this section is to show how we support reasoning about a TCS represented as a set of EC formulas in order to check its transactional consistency in two cases (see Figure 4): The first case is an a-priori checking using the transactional patterns consistency rule before running the TCS. The second case is an a-posteriori checking after TCS instance execution using TCS logs gathered in the set of termination states.

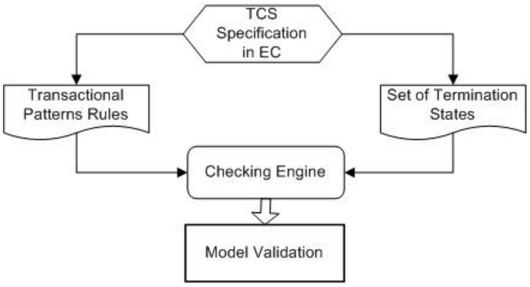


Fig. 4. Validation overview

4.1 A-Priori Checking

The need for a-priori verification is important for TCSs because they can be very complex processes, and therefore we need to check if the transactional behavior is consistent, which is not a trivial task as soon as a TCS process manages complex service dependencies. Indeed, TCS processes expect to enforce some high-level transactional policies which we have defined in a set of consistency transactional WS patterns rules. Our interest is to use these rules specified formally in EC to check process transactional consistency. Transforming TCS into EC predicates gives the opportunity to formalize these transactional policies by embedding logical predicates and to model-check if the TCS's transactional design complies with these policies, with respect to temporal constraints.

For instance, the designer can initially specify, as CS transactional behavior, that CCA and CRS will be compensated if CIC fails, CH is executed as alternative of CC failure, SH will be cancelled if CH fails, CA is pivot and SH is retriable. The EC formalisation of our motivating example (including the control and tehe transactional flow) is given in Table 3 using TCS patterns and transactional flow definitions from sections 2 and 3. Once it is rewritten using EC predicates, we propose to verify

Table 3. Example of EC formulas extracted from transactional WS patterns

Example's transactional WS patterns
1.*AND-split*(CRS, CIC, CCA)
2.*AND-join*(CIC, CCA, CA)
3.*XOR-split*(CA, CC, CH, SH)
4.$depAlt(CC, CH)$
5.$depCnl(CH, SH)$
6.$depCps(CIC, CCA)$
7.$depCps(CIC, CRS)$
8. CA^p
9. SH^r
EC formulas
1. $happens(CCA.activate(), t2) \wedge happens(CIC.activate(), t2) \leftarrow happens(CRS.complete(), t1) \wedge t1 < t2.$
2. $happens(CA.activate(), t3) \leftarrow happens(CCA.complete(), t1) \wedge happens(CIC.activate(), t2) \wedge t1 < t3 \wedge t2 < t3.$
3. $happens(CC.activate(), t2) \leftarrow happens(CA.complete(), t1) \wedge t1 < t2 \wedge happens(c_{CC}, t) \wedge \neg happens(c_{CH}, t) \wedge \neg happens(c_{SH}, t)$, $happens(CH.activate(), t2) \leftarrow happens(CA.complete(), t1) \wedge t1 < t2 \wedge happens(c_{CH}, t) \wedge \neg happens(c_{CC}, t) \wedge \neg happens(c_{SH}, t)$, $happens(SH.activate(), t2) \leftarrow happens(CA.complete(), t1) \wedge t1 < t2 \wedge happens(c_{SH}, t) \wedge \neg happens(c_{CH}, t) \wedge \neg happens(c_{CC}, t)$
4. $happens(CH.activate(), t2) \leftarrow happens(CC.fail(), t1) \wedge t1 < t2.$
5. $happens(SH.cancel(), t2) \leftarrow happens(CH.fail(), t1) \wedge t1 < t2.$
6. $happens(CCA.compensate(), t2) \leftarrow happens(CIC.fail(), t1) \wedge t1 < t2.$
7. $happens(CRS.compensate(), t2) \leftarrow happens(CIC.fail(), t1) \wedge t1 < t2.$
8. $\nexists(a() \in \mathcal{T}ransitions(CA^p) \mid (terminates(CA^p.a(), complete(CA^p), t) \leftarrow HoldsAt(complete(CA^p), t) \wedge Happens(CA^p.a(), t))).$
9. $Happens(SH^r.retry(), t) \leftarrow HoldsAt(failed(SH^r), t) \wedge t1 < t1.$

the designed transactional behavior consistency against the transactional constraints in Table 2.

For example, by checking the cancellation dependency between SH and CH reported in the predicates of line 5 in Table 3 against transactional consistency rules of the *XOR-split* pattern (rule 6), we observe an erroneous transactional dependency. These rules forbid cancellation dependencies between the composing services of the *XOR-split* pattern. Indeed, cancellation dependencies exist only between concurrent services, and in our example SH and CH are not concurrent. This basic example shows how it is possible to formally check and validate the consistency of TCS's transactional flow using the EC predicates.

4.2 A-Posteriori Checking

Our work attempts to apply Web service log-based analysis and process model checking techniques to provide knowledge about discrepancies between process models and related instances using a-posteriori verification. More precisely, given an event log, we want to verify a TCS's transactional properties after runtime, to provide knowledge about the context of and the reasons of discrepancies between process models and related instances.

This kind of verification is necessary since some interactions between Web services that constitute a process may be dynamically specified at runtime, causing unpredictable interactions with other services, and making the a-priori verification method insufficient as it only takes into account static aspects. To provide this verification, we use again logical predicates of the TCS model (see Table 3), but we compare these predicates

Table 4. WS Event log example

instance 1	instance 2	instance 3
$happens(CIC.fail(),1)$	$happens(CRS.complete(),8)$	$happens(CRS.complete(),30)$
$happens(CRS.compensate(),3)$	$happens(CCA.complete(),11)$	$happens(CRS.complete(),32)$
$happens(CCA.cancel(),4)$	$happens(CIC.complete(),12)$	$happens(CCA.complete(),33)$
$Initially_P(initial(CA))$	$happens(CA.complete(),18)$	$happens(CA.complete(),36)$
$Initially_P(initial(CC))$	$Initially_P(initial(CC))$	$happens(SH.fail(),42)$
$Initially_P(initial(SH))$	$happens(CH.fail(),19)$	$Initially_P(initial(CC))$
$Initially_P(initial(CH))$	$happens(SH.complete(),20)$	$Initially_P(initial(SH))$

with the events that occur during the process execution. When one or several predicates are unsatisfied, this means that we have wrong transactional behavior in the execution. Thus, it is possible to exactly pin down what happened.

In [10] it is demonstrated that the set of last or termination service state reports its transactional behavior. We distinguish two types of service termination states. The first one corresponds to the termination states reached after normal executions (without failures). The second kind of termination states corresponds to the ones reached in case of failure(s) of certain component service(s). Such a kind of termination states keeps track of failure(s) produced during the execution and the applied recovery mechanisms. For our a-posteriori verification approach, we propose to monitor only the final transition of each composing service that induces its termination state. Table 4 reports related final transitions of our motivating example. For instance, the predicate $happens(CIC.fail(),t)$ of instance 1 indicates that the termination state of the CIC is *failed*. This table reports respectively in the instances 1, 2 and 3 the failure of CIC, CH and SH services.

The key idea of the a-posteriori verification approach is to compare after the execution of a TCS instance its set of termination states (*i.e* table 4) to TCS's initially designed model to monitor whether it is coherent with the initial design and to detect potential discrepancies that can express incoherences or potential "new" process evolution requirements. For instance using the deduction algorithms in [10], we can deduce that the instance 1 in the table 4 reports two transactional dependencies after CIC failure: a compensation dependency from CIC to CRS and a cancellation dependency from CIC to CCA. Although the compensation dependency from CIC to CRS was reported in the initial design (see line 7 in Table 3), the cancellation dependency from CIC to CCA was not specified, only a compensation dependency from CIC to CCA is described. The designers predict only compensation dependency as recovery mechanism for CCA in case of CIC failure. But the compensation dependency can be applied only if the service state is completed. However, in this instance CCA is still executed as the only possible recovery mechanism applied was a cancellation dependency. Thus we can conclude through this monitoring phase that the designer miss to add this recovery mechanism. Similarly, the instance 2 in table 4 reports a "new" alternative dependency between CH and SH which is not reported by the initial design where designers do not predict CH failure. When such conditions occur, services monitoring has to fix this evolution requirement at runtime by updating the initial design.

In some cases, a recovery mechanism initially designed and a-priori verified can generate execution errors due unpredictable external factors (e.g. failures in the execution engine or system). For example, the instance 3 in Table 4 reports that our motivating example finishes its execution in incoherent way. In fact, after the failure of SH, the retriable property of this service which represents its recovery mechanism was not fulfilled. So the user should intervene to enforce the system to resume the execution by calling a *retry()* transition on SH until SH reaches *completed* as termination state.

As we have shown through this example, monitoring the "effective" transactional behavior allows us to detect design gaps and to improve the application reliability. Some deviations from the expected behavior may be highly desirable to detect process evolution and execution anomalies showing initially hardly foreseeable process parameters, constraints and needs.

5 Implementation

In this section, we describe the implementation work that we have done in order to validate our proposition. The first issue is related to the web service log collecting facilities. The present implementation uses the engine bpws4j[3] and log4j[4] to generate logging events. The choice of this engine was motivated by the fact that is a open source BPEL engine that can be customized to enable business recoverability and its use of Log4j that provides a robust, reliable, fully configurable, easily extendible, and easy to implement framework for logging Java applications for debugging and monitoring purposes. We have also specified regular expressions [11] to convert Log4j logging events to the required EC events.

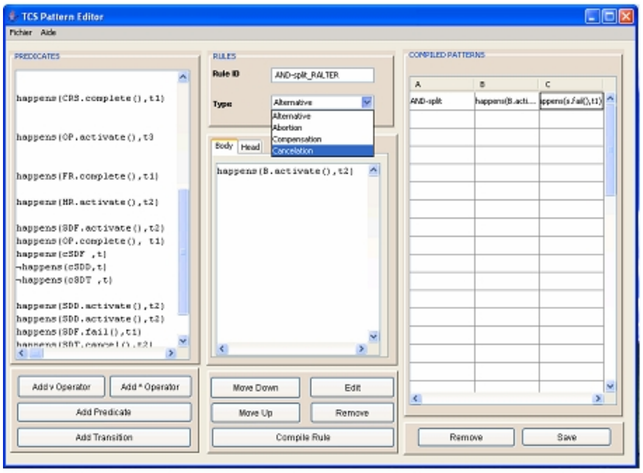
The transformation of a TCS model to its EC specification is built as a parser that can automatically transform a given set of transactional WS patterns into EC formulas according to the definitions and dependencies so far explained, and represented in an XML-based language that we have defined to represent EC formulas.

The patterns editor, shown in Figure 5, offers to service providers the different types of events and fluent initiation predicates that have been identified in the composition process and supports the specification of rules as logical combinations of these event and fluent initiation predicates. Service providers may also use the editor to define additional fluents or transitions to represent services, service states, and relevant initiation and holding predicates. When a pattern is specified, the editor can check its syntactic correctness.

The TCS consistency checker proposes a graphical user interface. This interface incorporates a tool that supports the specification of the TCS policies including a patterns editor that the user can use to specify the predicates and rules of each pattern and an interface of TCS consistency checker that displays the deviations from the specifications to check. A screenshot of the graphical interface of the TCS consistency checker is shown in Figure 6.

The user must load the TCS specification before the check consistency button can be enabled.Then he has to choose and select the patterns to be used. Following this, he

[3] http://alphaworks.ibm.com/tech/bpws4j
[4] http://logging.apache.org/log4j/docs/



Rule ID: AND-split_RALTER

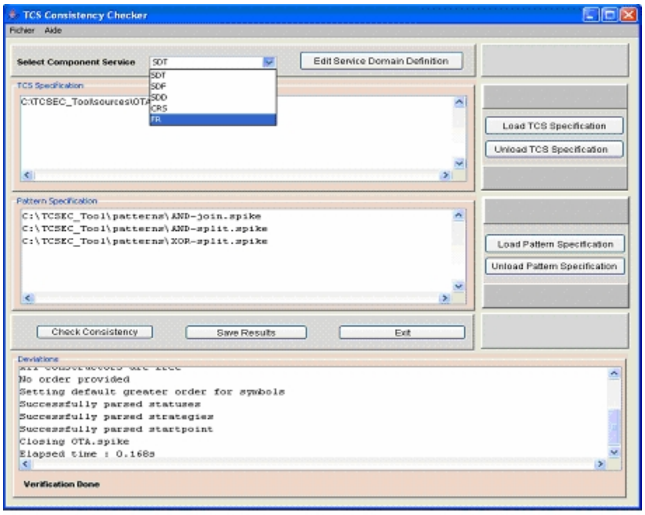
Type: Alternative

```
happens(CRS.complete(),t1)

happens(OP.activate(),t3

happens(FR.complete(),t1)

happens(MR.activate(),t2)

happens(SDF.activate(),t2)
happens(OP.complete(), t1)
happens(cSDF ,t)
¬happens(cSDD,t)
¬happens(cSDT ,t)

happens(SDD.activate(),t2)
happens(SDD.activate(),t2)
happens(SDF.fail(),t1)
```

```
happens(B.activate(),t2)
```

A	B	C
AND-split	happens(B.acti...	ppens(s.fail(),t1)

Fig. 5. A screenshot of the patterns editor



Select Component Service: SDT

```
C:\TCSEC_Tool\sources\OTA
```

```
C:\TCSEC_Tool\patterns\AND-join.spike
C:\TCSEC_Tool\patterns\AND-split.spike
C:\TCSEC_Tool\patterns\XOR-split.spike
```

```
No order provided
Setting default greater order for symbols
Successfully parsed statuses
Successfully parsed strategies
Successfully parsed startpoint
Closing OTA.spike
Elapsed time : 0.168s
```

Verification Done

Fig. 6. The graphical interface of the TCS consistency checker

can select component services and edit the domain definition of each one (transactional properties). When both the TCS specification and the patterns specifications are loaded, the verification process is ready to be executed. Results of the process verification can be saved in a file and therefore the deviations specifications can be analyzed and used to create queries to locate services that could substitute malfunctioning or unavailable services.

As The verification back-end, we have used the induction-based theorem prover SPIKE [12]. SPIKE was chosen for : (i) its high automation degree, (ii) its ability on case analysis, (iii) its refutational completeness (to find counter-examples), (iv) its incorporation of decision procedures (to automatically eliminate arithmetic tautologies produced during the proof attempt).

SPIKE proof method is based on cover set induction. In the first step, SPIKE encodes EC and TCS model ontology (Events, Fluents, Axioms, Log, Transactional WS patterns). Then, it computes induction variables to apply induction terms which basically represent all possible values that can be taken by the induction variables. Finally, it builds an algebraic specification from EC specification. with this specification, it can check all transactional properties by means of the powerful deductive techniques (rewriting and induction).

Given a conjecture (EC rule) to be checked, the prover selects induction variables and substitute them in all possible way by induction terms. This operation generates several instances of the conjecture which are then *simplified* by rules, lemmas, and induction hypotheses. Then, when SPIKE is called, either the consistency proof succeeds, or SPIKE 's proof-trace is used for extracting all scenarios which may lead to potential deviations. There are two possible scenarios. The first scenario is meaningless because conjectures are valid but it comes from a failed proof attempt of SPIKE . Such cases can be overcome by simply introducing new lemmas. The second one concerns cases corresponding to real deviations. The trace of SPIKE gives all necessary information (events, fluents and timepoints) to understand the origin of the inconsistency. Consequently, these information help designers to detect and manage transactional inconsistencies in a composite Web service.

6 Discussion

Generally, formal previous approaches develop, using their modeling formalisms, a set of techniques to analyze the composition model and check related properties. [13] proposes a formal framework for modeling, specifying and analyzing the global behavior of Web services compositions. This approach models web services by mealy machines (finite state machines with input an output). Based on this formal framework, the authors illustrate the unexpected nature of the interplay between local and global composite Web services. In [14], the authors propose a Petri net-based algebra for composing Web services. This formal model enables the verification of properties and the detection of inconsistencies both between and within services.

Although powerful, the above formal approaches may fail, in some cases, to ensure CS reliable executions even if they formally validate their composition models. This is because the properties specified in the studied composition models may not coincide with the reality (i.e., effective CSs executions). To the best of our knowledge, there are no approaches to transactional web services correction based on event-based logs, and in general there are very few contributions in this area. To deal with these issues, we described in this paper a combined a approach that describes a formal framework to check the transactional behavior of Web service composition before and after execution. Our approach provides a logical foundation to ensure transactional behavior consistency at design time and report recovery mechanisms deviations after runtime.

Firstly, we propose to formally specify Web service composition using transactional WS patterns (step 1). Transactional WS patterns can be seen as a convergence concept between workflow systems and transactional models to easily define flexible and reliable composite Web services by combining workflow flexibility and transactional processing reliability. Indeed, we can classify the current related Web service composition technologies in two classes, workflow based like WSBPEL [15] and WS-CDL [16] and transactional based like WS-AtomicTransaction [17], WS-BusinessActivity [18] and WS-TXM (Acid, BP, LRA) [19]. We can say that these technologies are standardized versions of the workflow approach or ATM adapted to work in a peer to peer environment. Consequently, they inherit the limitation of these two approaches: ensure reliability on behalf of process adequacy or the opposite. We believe that transactional patterns can complement these efforts.

Basically, WSBPEL and WS-CDL follow a workflow approach to define services compositions and services choreographies. Like workflow systems these two language meet the business process need in term of control structure. However, they are unable to ensure reliability especially according to the designers specific needs. Transactional WS patterns can be used on top of them to define reliable compositions. Then the defined model can be described either using WSBPEL or WS-CDL. WS-AtomicTransaction, WS-BusinessActivity and WS-TXM rely on ATM to define transactional coordination protocols. Like ATM these protocols are unable in most cases to model Business process due to their limited control structure. Transactional WS patterns allow to extend these protocols to support complex structure while preserving reliability. A composition of transactional WS patterns can be considered as a transactional protocol. Indeed, this approach allows for reliable, more complex, and more flexible compositions. In addition, it can coordinate services implemented with different technologies since we use only services transactional features (and not interested in how they are implemented).

The transactional WS patterns formally specified (using EC) as a set a logical formulas are used thereafter in our paper to support reasoning about a TCS to check its transactional consistency before (step 2) and after runtime (step 3). EC was chosen as an appropriate basis for formalising transactional composite Web services as both the properties and the transactional behaviour we are modelling are event driven. Compared to other formal languages, the EC Ontology offers a complete and efficient formal support for checking CS transactional behavior. Indeed, the types considered (fluents, events, time points) are enough to express all the data used for WS composition [20]. In addition the language includes a number of base predicates ($initiates$, $terminates$, $holdsAt$, $happens$, ...) which are used to define some auxiliary predicates; and domain independent axioms. These axioms are very important to reduce the computational complexity of the proof procedure

In a-priori checking approach (step 2), transactional WS patterns are initially extracted from TCS specification and transactional consistency rules are defined in EC. These rules are used in an EC checker to ensure model transactional consistency. In the a-posteriori checking approach (step 3), the main focus has put on verification. This means that given the designed TCS and the collected event log, we check whether the observed behavior matches the (un)expected/(un)desirable transactional behavior to ensure service execution reliability. EC supports deductive, inductive and abdicative reasoning. Deduction uses the description of the process behaviour to derive the fluents that will hold at a particular point in time. The deduction approach was used in our a-priori checking approach. Given the descriptions of the behaviour of the model, abduction can be used to determine the sequence of events that need to occur such that a given set of fluents will hold at a specified point in time. Therefore, it is possible to detect conflicts when the applicability of the properties is constrained on the runtime state of the system and the analysis can be performed even with partial specifications of the system state. The deduction approach was used in our a-posteriori checking approach. Finally, induction aims to derive the descriptions of the process behaviour from a given event history and information about the fluents that hold at different points of time. This skill is used actually in our current work [11] to enable the verification of behavioral properties in web service composition using findings in the fields of process mining.

In our future work, we are working on providing a tool which uses the recorded deviations to generate queries for discovering services that could substitute for malfunctioning services. We are also trying to enhance the capabilities of the transactional behavior monitoring techniques by enriching TCS logs and extracting data flow dependencies. We aim also to adapt/combine workflow mining techniques to the web services related fields. A first work was validated in [21,22] where we applied mining techniques to discover and improve composite Web service transactional behavior. We are also working to discover more complex patterns by enriching collected TCS log by Data flow information for instance.

References

1. Kowalski, R., Sergot, M.J.: A logic-based calculus of events. New generation Computing 4(1), 67–95 (1986)
2. Elmagarmid, A.K. (ed.): Database transaction models for advanced applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1992)
3. van der Aalst, W.M.P., van Hee, K.M.: Workflow Management: models, methods and tools. In: Cooperative Information Systems, MIT Press, Cambridge (2002)
4. Mueller, E.T.: Event calculus reasoning through satisfiability. J. Log. and Comput. 14(5), 703–730 (2004)
5. Mehrotra, S., Rastogi, R., Korth, H.F., Silberschatz, A.: A transaction model for multidatabase systems. In: ICDCS, pp. 56–63 (1992)
6. Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A.H.H., Elmagarmid, A.K.: Business-to-business interactions: issues and enabling technologies. The VLDB Journal 12(1), 59–85 (2003)
7. van der Aalst, W.M.P., Barros, A.P., ter Hofstede, A.H.M., Kiepuszewski, B.: Advanced Workflow Patterns. In: Scheuermann, P., Etzion, O. (eds.) CoopIS 2000. LNCS, vol. 1901, pp. 18–29. Springer, Heidelberg (2000)
8. van der Aalst, W.M.P., ter Hofstede, A.H.M.: Yawl: yet another workflow language. Inf. Syst. 30(4), 245–275 (2005)
9. Bhiri, S., Godart, C., Perrin, O.: Transactional patterns for reliable web services compositions. In: Wolber, D., Calder, N., Brooks, C., Ginige, A. (eds.) ICWE, pp. 137–144. ACM, New York (2006)
10. Bhiri, S., Perrin, O., Godart, C.: Ensuring required failure atomicity of composite web services. In: WWW, pp. 138–147 (2005)

11. Rouached, M., Gaaloul, W., van der, A.W.M.P., Bhiri, S., Godart, C.: Web service mining and verification of properties: An approach based on event calculus. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 408–425. Springer, Heidelberg (2006)
12. Stratulat, S.: A general framework to build contextual cover set induction provers. Journal of Symbolic Computation 32(4), 403–445 (2001)
13. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: WWW, pp. 403–410 (2003)
14. Hamadi, R., Benatallah, B.: A petri net-based model for web service composition. In: ADCDT, pp. 191–200 (2003)
15. IBM BEA and Microsoft. Business process execution language for web services (bpel4ws) (2003)
16. Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web services choreography description language version 1.0. (2004), http://www.w3.org/TR/ws-cdl-10
17. Cabrera, L.F., et.al.: Web services atomic transaction (ws-atomictransaction) (September 2003)
18. Cabrera, L.F., et al.: Web services business activity framework (ws-businessactivity) (January 2004)
19. Arjuna, Fujitsu, IONA, Oracle, and Sun. Web services composite application framework (ws-caf), http://www.arjuna.com/standards/ws-caf/
20. Rouached, M., Perrin, O., Godart, C.: Towards formal verification of web service composition. In: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, pp. 257–273. Springer, Heidelberg (2006)
21. Bhiri, S., Gaaloul, W., Godart, C.: Discovering and improving recovery mechanisms of compositeweb services. In: ICWS, pp. 99–110. IEEE Computer Society, Los Alamitos (2006)
22. Gaaloul, W., Bhiri, S., Haller, A.: Mining and re-engineering transactional workflows for reliable executions. In: ER 2007. 26th International Conference on Conceptual Modeling (November 2007)

Matching Cognitive Characteristics of Actors and Tasks

S.J. Overbeek[1], P. van Bommel[2], H.A. (Erik) Proper[2], and D.B.B. Rijsenbrij[2]

[1] e-office B.V., Duwboot 20, 3991 CD Houten, The Netherlands, EU
Sietse.Overbeek@e-office.com

[2] Institute for Computing and Information Sciences, Radboud University Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, EU
{P.vanBommel,E.Proper,D.Rijsenbrij}@cs.ru.nl

Abstract. Acquisition, application and testing of knowledge by actors trying to fulfill knowledge intensive tasks is becoming increasingly important for organizations due to trends such as globalization, the emergence of virtual organizations and growing product complexity. An actor's management of basic cognitive functions, however, is at stake because of this increase in the need to acquire, apply and test knowledge during daily work. This paper specifically focusses on matchmaking between the cognitive characteristics supplied by an actor and the cognitive characteristics required to fulfill a certain knowledge intensive task. This is based on a categorization and characterization of actors and knowledge intensive tasks. A framework for a cognitive matchmaker system is introduced to compute actual match values and to be able to reason about the suitability of a specific actor to fulfill a task of a certain type.

1 Introduction

The importance of an actor's abilities to acquire, apply and test already applied knowledge increases due to e.g. growing product complexity, the move toward globalization, the emergence of virtual organizations and the increase in focus on customer orientation [1]. A knowledge intensive task is a task for which acquisition, application or testing of knowledge is necessary in order to successfully fulfill the task. When the pressure to acquire, apply and test more knowledge increases, actors struggle to manage their basic cognitive functions like e.g. the willpower to fulfill a task or maintaining awareness of the requirements to fulfill a task. These cognitive functions are also referred to as *volition* and *sentience* respectively in cognitive literature [2,3]. Difficulties to control basic cognitive functions influences practice and potentially threatens the success of task fulfillment [4]. Research in cognitive psychology has demonstrated that individual knowledge processing is negatively influenced when experiencing an overload of knowledge that needs to be processed. A burden of knowledge processing events may cause actors to underrate the rate of events [5] and to be overconfident [6].

In [7] we have discussed several types of knowledge intensive tasks, each characterized by their characteristics. These task types consist of an *acquisition* task, a *synthesis* task and a *testing* task. An acquisition task is related with the elicitation of knowledge. A synthesis task is related with the actual utilization of the acquired knowledge. Lastly, a testing task is related with the identification and application of knowledge in practice inducing an improvement of the specific knowledge applied. The characteristics belonging to each task type indicate the cognitive requirements necessary for an actor to successfully fulfill an instance of a specific task type. Based on this earlier work, the research reported in this paper is specifically concerned with the *matching* of cognitive characteristics required to fulfill a certain task instance with the cognitive characteristics actually *possessed* by an actor. The ambition of this paper, however, is not to come up with a tool that will be concerned with cognitive matchmaking. Instead, the emphasis is on developing a framework which includes the aspects of such a matchmaking process and to acquire insight in how these aspects can be tackled.

2 Cognitive Actor Settings

Before elaborating on matching cognitive characteristics possessed by an actor with the cognitive characteristics required when fulfilling a task instance, a characterization of possible actor types is needed.

2.1 Actor Types

Actor types may draw from a pool of basic cognitive characteristics an actor might possess, such as sentience, volition and causability [8]. No one actor type necessarily has all of these characteristics and some have more than others. Using a series of linguistic diagnostics, Dowty [8] has shown that each of these characteristics can be isolated from the others and so should be treated as distinct. The following characteristics can thus be distinguished that can be utilized to generate cognitive settings of possible different actor types.

The *volition* characteristic is concerned with an actor's willpower to fulfill some knowledge intensive task instance. *Sentience* expresses that an actor has complete awareness of required knowledge to fulfill some task instance. The *causability* characteristic expresses that an actor has the ability to exert an influence on state changes of knowledge involved during fulfillment of some task instance. During fulfillment of certain knowledge intensive task instances an actor should be able to improve its own cognitive abilities. This is indicated by the *improvability* characteristic. The *independency* characteristic is necessary to be able to determine if an actor is able to fulfill some task instance on its own.

Having determined possible cognitive characteristics an actor may have it is now appropriate to distinguish several actor types. The combination of an actor type with the cognitive characteristics belonging to a type forms a *cognitive actor setting*. This characterization is shown in table 1. The five distinguished actor types are based on a classification of knowledge worker types [9] and on linguistic literature [2]. The set of actor types can be represented as:

$$\{\mathtt{experiencer}, \mathtt{collaborator}, \mathtt{expert}, \mathtt{integrator}, \mathtt{transactor}\} \subseteq \mathcal{AT} \qquad (1)$$

Table 1. Cognitive actor settings characterized

	$\mathcal{CC}$				
$\mathcal{AT}$	Volition	Sentience	Causability	Improvability	Independency
Experiencer	–	×	–	–	–
Collaborator	×	–	×	×	–
Expert	×	×	×	×	×
Integrator	×	–	×	–	–
Transactor	×	×	–	–	×

The set of cognitive characteristics can be represented as:

$$\{\texttt{volition}, \texttt{sentience}, \texttt{causability}, \texttt{improvability}, \texttt{independency}\} \subseteq \mathcal{CC} \quad (2)$$

An important remark to make here is that the actor types as well as the cognitive characteristics are not limited to five actor types and five cognitive characteristics. However, in this paper we restrict ourselves to the above mutually independent cognitive actor settings. The actor types as shown in table 1 can now be introduced.

The *experiencer* actor type has the sentience characteristic only. An experiencer is thus only aware of all the knowledge requirements to fulfill some task instance. Consider for example the following sentence: *John thoroughly reads an article about balanced scorecards before joining a meeting about balanced scorecards*. This indicates that John, as an experiencer, probably understands that reading an article about balanced scorecards is enough to successfully prepare himself for a meeting about that topic. The *collaborator* actor type possesses the volition, causability and improvability characteristics. A collaborator has the ability to exert an influence on state changes of knowledge involved during fulfillment of a task instance. During fulfillment of a knowledge intensive task instance a collaborator is also able to improve its own cognitive abilities. However, a collaborator does not have complete awareness of all required knowledge to fulfill a task instance and requires others to fulfill a task instance. Consider the following example: *John works at a hospital and requires knowledge about a patient's history. Therefore, he acquires the most recent patient log from a colleague*. This indicates that John, as a collaborator, understands that in order to acquire knowledge about a patient's history he must collaborate with another actor. After that John is able to update the patient's log with recent changes. An *expert* possesses all characteristics depicted in table 1. Suppose that John is an assistant professor working at a university and he would like to solve a difficult mathematical problem when developing a theory. He then uses his own knowledge about mathematics to solve the problem. John is also able to combine and modify his own knowledge while solving the problem and he can also learn from that. An *integrator* is able to fulfill a knowledge intensive task instance by working together and is able to initiate state changes of knowledge involved during task instance fulfillment. An integrator primarily wishes to acquire and apply knowledge of the highest possible quality. An engineer contributing to the construction of a flood barrier is an example of an integrator. Volition, sentience and independency are the characteristics belonging to the *transactor* actor type. A transactor can fulfill a task instance without collaborating with others and is not required to cause modifications in the knowledge acquired and applied during task fulfillment. A customer support employee working at a software company is an example of a transactor.

A specific instantiation of an actor type is expressed by $\mathsf{AType} : \mathcal{AI} \rightarrow \mathcal{AT}$, where $\mathcal{AI}$ is a set of *actor instances*. The example $\mathsf{AType}(a) = \texttt{experiencer}$ for instance expresses that an actor a can be classified as an experiencer. We can view the actor that is specifically fulfilling a task instance $i \in \mathcal{TI}$ as a function $\mathsf{Fulfillment} : \mathcal{AI} \rightarrow \wp(\mathcal{TI})$. Here, $\mathcal{TI}$ is a set of task instances. An actor a that fulfills a task instance i can be expressed as $\mathsf{Fulfillment}(a) = \{i\}$. A specific instantiation of a task type is expressed by $\mathsf{TType} : \mathcal{TI} \rightarrow \mathcal{TT}$, where $\mathcal{TT}$ is a set of *task types* that can be instantiated by a specific task instance. The expression $\mathsf{TType}(i) = \texttt{acquisition}$ can be used to assert that a task instance i is characterized as an acquisition task.

3 Cognitive Matchmaker System

In this section a framework for a cognitive matchmaker system is introduced that is able to compute a match between cognitive characteristics required for a specific task type and cognitive characteristics that are provided by a specific actor type. As a running example, we use the matchmaker system to match the cognitive characteristics offered by the *transactor* actor type with the required cognitive characteristics of a *synthesis* task. Figure 1 shows the architecture of the system on a conceptual level, which is translated into the formalisms throughout this section. In section 2, a function $\mathsf{AChar}_j(a) = C$ indicated the cognitive

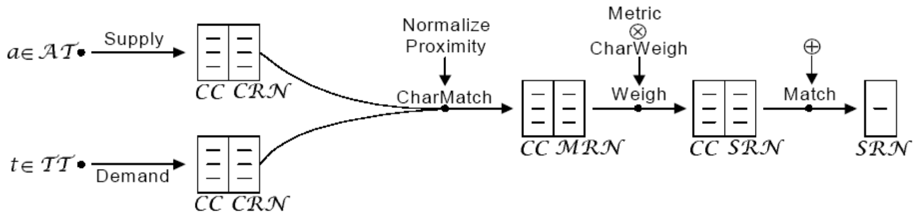


Fig. 1. Cognitive matchmaker system

characteristics that characterized an actor instance of a certain type, where j is a task type belonging to the set of task types $\mathcal{TT}$, a is an actor instance belonging to the set of actor instances $\mathcal{AI}$ and C is a set of cognitive characteristics that is a subset of or equal to $\mathcal{CC}$. Recall from section 2 that the corresponding actor type can be found by using the *actor type* function: $\mathsf{AType}(a) = j$. With this in mind, a *supply* function can be modeled that returns a value expressing to what extent an actor type offers a certain cognitive characteristic:

$$\mathsf{Supply} : \mathcal{AT} \rightarrow (\mathcal{CC} \rightarrow \mathcal{CRN}) \tag{3}$$

The expression $\mathsf{Supply}_{\mathtt{transactor}}(\mathsf{s}) = 10$ shows that an actor characterized by the transactor type offers the sentience characteristic and is at least capable to perform this characteristic at level 10. Note that for readability reasons the word 'sentience' has been abbreviated to the letter 's'. The resulting value '10' is part of a characteristic rank domain $\mathcal{CRN}$ which contains integer values within the range $[0, 10]$. The hard values as part of a domain of values can be found using the following function:

$$\mathsf{Numerical} : \wp(\mathcal{RN}) \rightarrow \mathbb{R} \tag{4}$$

Here, the set $\mathcal{RN}$ contains rank values and $\mathcal{CRN} \subseteq \mathcal{RN}$. Formally, the characteristic rank domain includes the following hard values: $\mathsf{Numerical}(\mathcal{CRN}) = [0, 10]$. A value of 0 means that an actor is not able to offer a certain characteristic, a value of 5 means that an actor is able to offer a characteristic at an average level and a value of 10 means that an actor is able to offer a characteristic at the highest level. So, in the case of the example, the transactor is able to offer the sentience characteristic at the highest level.

Besides modeling a supply function, a demand function is needed that returns a value expressing to what extent a cognitive characteristic is *required* for a certain task type:

$$\mathsf{Demand} : \mathcal{TT} \rightarrow (\mathcal{CC} \rightarrow \mathcal{CRN}) \tag{5}$$

The expression $\mathsf{Demand}_{\mathtt{synthesis}}(\mathsf{s}) = 10$ indicates that a sentience characteristic is required at the highest level in order to fulfill a synthesis task. The supply and demand functions can now be utilized to compute the characteristic match.

3.1 Characteristic Match

In this section, a characteristic match function is defined to compare the resulting values from the supply and demand functions. This comparison provides insight in the way supply and demand of cognitive characteristics are matched with each other. In order to model a *characteristic match* function, an actor type as well as a task type are required as input, together with a cognitive characteristic:

$$\mathsf{CharMatch} : \mathcal{AT} \times \mathcal{TT} \rightarrow (\mathcal{CC} \rightarrow \mathcal{MRN}) \tag{6}$$

As can be seen in figure 1, the characteristic match function returns a value from the match rank domain, where $\mathcal{MRN} \subseteq \mathcal{RN}$. The match rank domain includes the following values: $\mathsf{Numerical}(\mathcal{MRN}) = [0, 10]$.

To compute the actual characteristic match value, a *proximity* function is necessary to be able to define the characteristic match function. This proximity function computes the proximity of the level an actor offers a certain cognitive characteristic related to the level that is required in order to fulfill a task of a certain type. The values that are used as input for the proximity function are part of the characteristic rank domain. The resulting proximity value is then a value that is part of the match rank domain:

$$\mathsf{Proximity} : \mathcal{CRN} \times \mathcal{CRN} \rightarrow \mathcal{MRN} \tag{7}$$

A normalization function can be introduced that calculates the numerical proximity of demand and supply when a cognitive characteristic is concerned:

$$\mathsf{Normalize} : \mathbb{R} \rightarrow [0,1] \tag{8}$$

The normalization function can be defined by using the supply and demand functions and two additional constants `min` and `max`:

$$\mathsf{Normalize}(\mathsf{Supply}_i(c) - \mathsf{Demand}_j(c)) \triangleq \frac{\mathsf{Supply}_i(c) - \mathsf{Demand}_j(c) + \mathtt{max} - \mathtt{min}}{2 \cdot (\mathtt{max} - \mathtt{min})} \tag{9}$$

Here, i is an actor type of the set $\mathcal{AT}$, j is a task type of the set $\mathcal{TT}$ and c is a cognitive characteristic of the set $\mathcal{CC}$. The values of the constants `min` and `max` can be determined by interpreting the minimum and the maximum values of a specific ranking domain. So, in the case of the running example $\mathtt{min} = 0$ and $\mathtt{max} = 10$ when the characteristic rank domain is concerned. The minimum value that can be returned by the normalization function is 0. This occurs if there is absolutely no supply (i.e. an incapable actor is concerned) but there is a maximum demand of a certain cognitive characteristic in order to fulfill a task of a certain type. This situation is depicted below:

$$\mathsf{Normalize}(0 - 10) = \frac{0 - 10 + \mathtt{max} - \mathtt{min}}{2 \cdot (\mathtt{max} - \mathtt{min})} = 0$$

In the case of an overqualified actor that is more capable to perform a cognitive characteristic than is required, the normalization function returns 1:

$$\mathsf{Normalize}(10 - 0) = \frac{10 - 0 + \mathtt{max} - \mathtt{min}}{2 \cdot (\mathtt{max} - \mathtt{min})} = 1$$

This means that the normalization function normalizes the proximity of supply and demand between 0 and 1. Using the normalization function, the proximity function can now be defined as follows:

$$\mathsf{Proximity}(\mathsf{Supply}_i(c), \mathsf{Demand}_j(c)) \triangleq \mathsf{Normalize}(\mathsf{Supply}_i(c) - \mathsf{Demand}_j(c)) \tag{10}$$

For the running example the proximity function as defined above results in:

$$\mathsf{Proximity}(10, 10) = \mathsf{Normalize}(10 - 10) = 0.5$$

Now with the introduction of a proximity function the characteristic match can be defined by computing the proximity of demand and supply in the context of a given characteristic:

$$\mathsf{CharMatch}(i, j) \triangleq \lambda_{c \in \mathcal{CC}} \cdot \mathsf{Proximity}(\mathsf{Supply}_i(c), \mathsf{Demand}_j(c)) \tag{11}$$

Recall from section 3 that an actor of the transactor type is able to perform the sentience characteristic at level 10, which equals the level to what extent a sentience characteristic should be offered for a synthesis task type. In the case of our example the characteristic match results in:

$$\begin{aligned} &\mathsf{CharMatch}(\mathtt{transactor}, \mathtt{synthesis}) = \\ &\quad \lambda_{\mathsf{s} \in \mathcal{CC}} \cdot \mathsf{Proximity}(\mathsf{Supply}_{\mathtt{transactor}}(\mathsf{s}), \mathsf{Demand}_{\mathtt{synthesis}}(\mathsf{s})) = \\ &\quad \mathsf{Proximity}(10, 10) = 0.5 \end{aligned}$$

This example shows that for the transactor / synthesis task combination the eventual *proximity value* is 0.5. However, this proximity value is only related to the demand and supply of one specific cognitive characteristic. To compute a total match of the required cognitive characteristics for a task type and the characteristics offered, a *weighed suitability match* can now be introduced.

3.2 Weighed Suitability Match

The cognitive matchmaker system is completed by introducing a weighed suitability match, as is shown in the rightmost part of figure 1:

$$\mathsf{Match} : \mathcal{AT} \times \mathcal{TT} \rightarrow \mathcal{SRN} \tag{12}$$

This function returns a value from the suitability rank domain, where $\mathcal{SRN} \subseteq \mathcal{RN}$. The suitability rank domain includes the following values: $\mathsf{Numerical}(\mathcal{SRN}) = [0, 10]$. This means that an actor of a certain type can have suitability levels ranging from 0 to 10. To determine the suitability of the transactor fulfilling the synthesis task, the calculated proximity of demand and supply of a cognitive characteristic $c \in \mathcal{CC}$ can be weighed:

$$\mathsf{Weigh} : (\mathcal{CC} \rightarrow \mathcal{MRN}) \rightarrow (\mathcal{CC} \rightarrow \mathcal{SRN}) \tag{13}$$

To define the weigh function several other functions are necessary, though. As can be seen in figure 1 the weigh function uses the input from the characteristic match function and returns a value from the suitability rank domain as output. To construct the weigh function, a function is needed that has a match rank metric (i.e. the proximity value) as its input and a suitability rank metric as its output:

$$\mathsf{Metric} : \mathcal{MRN} \rightarrow \mathcal{SRN} \tag{14}$$

For instance, $\mathsf{Metric}(0.5) = 0.5$ shows that the value 0.5, which is the proximity value, equals the value 0.5 which is a suitability rank metric. A characteristic weigh function is needed to actually weigh the importance of a certain cognitive characteristic to fulfill a task of a certain type:

$$\mathsf{CharWeigh} : \mathcal{CC} \rightarrow \mathcal{SRN} \tag{15}$$

So, $\mathsf{CharWeigh}(\mathsf{s}) = 1.5$ means that a weigh factor of 1.5 is given to indicate the importance to offer the sentience cognitive characteristic (for a certain task). Finally, the $\otimes$ operator is also needed to define a definite weigh function:

$$\otimes : \mathcal{SRN} \times \mathcal{SRN} \rightarrow \mathcal{SRN} \tag{16}$$

The $\otimes$ operator is necessary to multiply the metric value with the characteristic weigh value. Multiplying the values mentioned above results in: $0.5 \otimes 1.5 = 0.75$. The weigh function can now be defined as:

$$\mathsf{Weigh}(c, \mathsf{CharMatch}(i, j)) \triangleq \lambda_{c \in \mathcal{CC}} \cdot \mathsf{Metric}(\mathsf{CharMatch}(i, j)) \otimes \mathsf{CharWeigh}(c) \tag{17}$$

Here, $c \in \mathcal{CC}, i \in \mathcal{AT}$ and $j \in \mathcal{TT}$. Continuing the running example, we would like to calculate the suitability of the transactor that is fulfilling a synthesis task. Considering the sentience characteristic only, this can be computed as follows:

$$\begin{aligned}
&\mathsf{Weigh}(\mathsf{s}, \mathsf{CharMatch}(\mathtt{transactor}, \mathtt{synthesis})) = \\
&\quad \lambda_{\mathsf{s} \in \mathcal{CC}} \cdot \mathsf{Metric}(0.5) \otimes \mathsf{CharWeigh}(\mathsf{s}) = \\
&\quad 0.5 \otimes 1.5 = 0.75
\end{aligned}$$

In order to calculate the suitability match of the transactor related to the synthesis task, it is mandatory to determine the cognitive characteristics supplied by the actor and demanded by the task. The transactor actor type supplies the *volition*, *sentience* and *independency* characteristics as is shown in table 1. The synthesis task type can be characterized by the *applicability* and *correctness* characteristics [7]. These characteristics are explained as follows. An actor should provide the applicability characteristic to be able to apply knowledge during task fulfillment and to make sure that the applied knowledge has a useful effect on successfully completing the task. An actor should provide the correctness characteristic to be able to judge the usefulness of applied knowledge in a task and to be sure that applied knowledge meets its requirements.

The set $\mathcal{CC}$ contains the following characteristics in the case of the running example: $\{\texttt{volition}, \texttt{sentience}, \texttt{independency}, \texttt{applicability}, \texttt{correctness}\} \subseteq \mathcal{CC}$. For all these characteristics a weigh value needs to be determined as in the example expression of function 17. This is necessary to compute a final *suitability match* resulting in one suitability rank value. Assume that the actual characteristic weigh values (each assigned to one cognitive characteristic as part of the set $\mathcal{CC}$) are: 2, 1.5, 0.5, 3 and 3. Note that these characteristic weigh values always summate to one and the same total value. In the case of our example the characteristic weigh values summate to 10. Thus, no matter how the weigh values are divided across the cognitive characteristics, they should always summate to a total of 10.

The results of the weighed characteristic matches have to be summated to generate a single *suitability match* value. To summate these values a $\oplus$ operator is required:

$$\oplus : \mathcal{SRN} \times \mathcal{SRN} \rightarrow \mathcal{SRN} \tag{18}$$

Now the final match function can be defined using the aforementioned functions:

$$\mathsf{Match}(i, j) \triangleq \bigoplus_{c \in \mathcal{CC}} \mathsf{Weigh}(c, \mathsf{CharMatch}(i, j)) \tag{19}$$

In the match function $i \in \mathcal{AT}, c \in \mathcal{CC}$ and $j \in \mathcal{TT}$. For the running example this means that the suitability match value of the transactor fulfilling a task instance of the synthesis type is computed as follows:

$$\mathsf{Match}(\texttt{transactor}, \texttt{synthesis}) = 1 \oplus 0.75 \oplus 0.5 \oplus 0.75 \oplus 0.75 = \mathbf{3.75}$$

As a result of the suitability match it can be concluded that the suitability of an actor characterized by the transactor type fulfilling a task instance of the synthesis type is 3.75. Remember that the lowest suitability value is 0 and the highest suitability value that can be reached is 10. The lowest value is reached if the supply of every characteristic is 0 and the demand of every characteristic is 10. The highest value is reached in the case of complete overqualification, i.e. if the supply of every characteristic is 10 and the demand of every characteristic is 0. At this point a decision can be made whether or not the specific actor is suitable enough to fulfill this task or if another actor is present who should be more suitable, i.e. has a better suitability match value. The suitability of

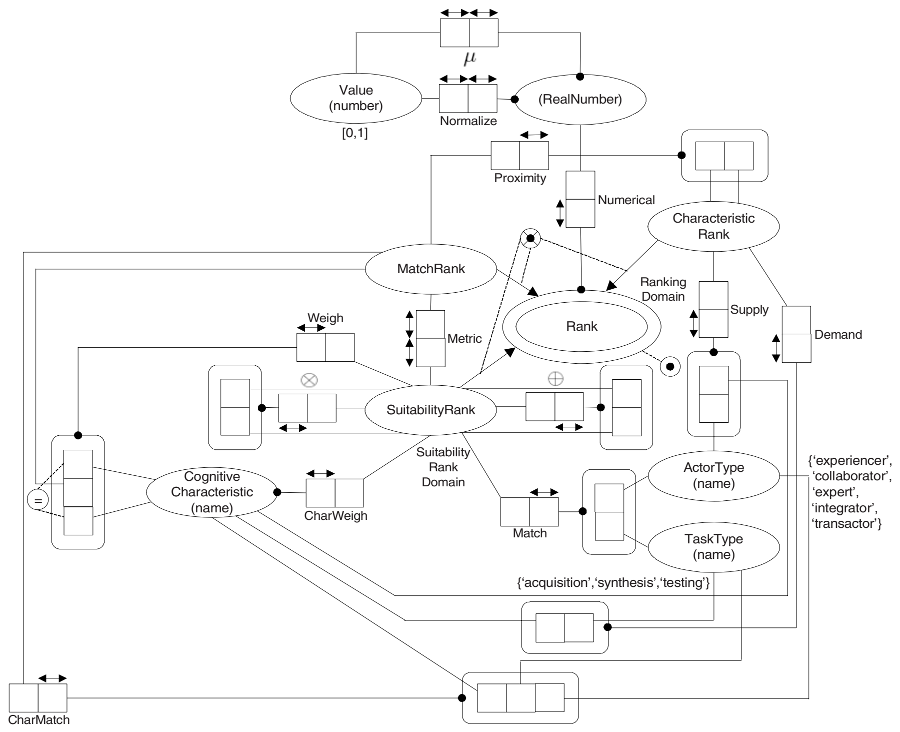


Fig. 2. Object-Role Modeling (ORM) model of the cognitive matchmaker system

an actor to fulfill a certain task is best if the resulting suitability value is 5. Underqualification as well as overqualification are both considered undesirable.

A certainty function can now be introduced to make sure how certain it is that an actor is suitable to fulfill a task:

$$\mu : \mathbb{R} \rightarrow [0,1] \tag{20}$$

A linear certainty function can be defined as follows:

$$\mu(u) \triangleq \begin{cases} \frac{2}{\texttt{min}+\texttt{max}} \cdot u & \texttt{min} \leq u \leq \frac{\texttt{min}+\texttt{max}}{2} \\ \frac{-2}{\texttt{min}+\texttt{max}} \cdot u + 2 & \frac{\texttt{min}+\texttt{max}}{2} \leq u \leq \texttt{max} \end{cases} \tag{21}$$

For the running example, where $\texttt{min} = 0$ and $\texttt{max} = 10$, the following expression shows that the certainty that the transactor is suitable to fulfill the synthesis task is 0.75:

$$\mu(3.75) = \frac{2}{0+10} \cdot 3.75 = 0.75$$

This can be interpreted as being 75% sure that the transactor is suitable enough to fulfill the synthesis task. It might be a good choice to let the transactor fulfill the synthesis task, unless an available actor characterized by another type provides a better match. In order to also have a graphical representation of the discussed definitions throughout section 3, an Object-Role Modeling (ORM) model is presented in figure 2. For details on Object-Role Modeling, see e.g. [10].

4 Conclusion

This paper describes a categorization and characterization of actors that are able to fulfill knowledge intensive tasks, illustrated by cognitive characteristics indicating actor abilities for task fulfillment. Proceeding from these characteristics a running example, in which a match is determined of an actor characterized by the transactor type wishing to fulfill a synthesis task, shows how the theory can be materialized.

References

1. Staab, S., Studer, R., Schnurr, H.P., Sure, Y.: Knowledge processes and ontologies. IEEE Intelligent Systems 16(1), 26–34 (2001)
2. Kako, E.: Thematic role properties of subjects and objects. Cognition 101(1), 1–42 (2006)
3. Weir, C.R., Nebeker, J., Bret, L., Campo, R., Drews, F., LeBar, B.: A cognitive task analysis of information management strategies in a computerized provider order entry environment. Journal of the American Medical Informatics Association 14(1), 65–75 (2007)
4. Meiran, N.: Modeling cognitive control in task-switching. Psychological Research 63(3–4), 234–249 (2000)
5. Hertwig, R., Barron, G., Weber, E., Erev, I.: The role of information sampling in risky choice. In: Fiedler, K., Juslin, P. (eds.) Information Sampling and Adaptive Cognition, pp. 72–91. Cambridge University Press, New York, NY, USA (2006)
6. Koehler, D.: Explanation, imagination, and confidence in judgment. Psychological Bulletin 110(3), 499–519 (1991)
7. Overbeek, S., van Bommel, P., Proper, H., Rijsenbrij, D.: Characterizing knowledge intensive tasks indicating cognitive requirements - Scenarios in methods for specific tasks. In: Ralyt, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Proceedings of the IFIP TC8/WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences, Geneva, Switzerland. IFIP, vol. 244, pp. 100–114. Springer, Boston, USA (2007)
8. Dowty, D.: Thematic proto-roles and argument selection. Language 67(3), 547–619 (1991)
9. Davenport, T.: Thinking for a Living – How to get Better Performances and Results from Knowledge Workers. Harvard Business School Press, Boston, MA, USA (2005)
10. Halpin, T.: Information Modeling and Relational Databases, from Conceptual Analysis to Logical Design. Morgan Kaufmann, San Mateo, CA, USA (2001)

The OpenKnowledge System: An Interaction-Centered Approach to Knowledge Sharing

Ronny Siebes[1], Dave Dupplaw[2], Spyros Kotoulas[1], Adrian Perreau de Pinninck[3], Frank van Harmelen[1], and David Robertson[4]

[1] Vrije Universiteit Amsterdam
{ronny,kot,frankh}@few.vu.nl
[2] University of Southampton, UK
dpd@ecs.soton.ac.uk
[3] Artificial Intelligence Research Institute (IIIA - CSIC)
adrianp@iiia.csic.es
[4] The University of Edinburgh, Edinburgh, UK
dr@inf.ed.ac.uk

Abstract. The information that is made available through the semantic web will be accessed through complex programs (web-services, sensors, *etc.*) that may interact in sophisticated ways. Composition guided simply by the specifications of programs' inputs and outputs is insufficient to obtain reliable aggregate performance - hence the recognised need for process models to specify the interactions required between programs. These interaction models, however, are traditionally viewed as a *consequence* of service composition rather than as the focal point for *facilitating* composition. We describe an operational system that uses models of interaction as the focus for knowledge exchange. Our implementation adopts a peer to peer architecture, thus making minimal assumptions about centralisation of knowledge sources, discovery and interaction control.

1 Introduction

The pool of potentially available knowledge on the Internet is immeasurably large. It is fed by the traditional Web: by application programs feeding data onto the Web, by Web services accessed through various forms of application interface, by devices that sense the physical environment, and so on. It is consumed in a wide variety of ways and by diverse mechanisms (and of course consumers may also be suppliers). The aspiration of OpenKnowledge is to allow knowledge to be shared freely and reliably, regardless of the source or consumer. Reliability here is interpreted as a semantic issue. The Internet is in the fortunate situation that physical and syntactic reliability have been solved to satisfactory degrees, making semantic reliability the main challenge. Semantic reliability means that we want the meaning ascribed to knowledge that is fed into the pool, to be preserved adequately for the purposes of consumers.

Of course such "open knowledge sharing" is an aspiration that we know to be unattainable, in the strong sense where all knowledge supplied can be consumed with perfect freedom and reliability. Globally consistent common knowledge is impossible to guarantee in an asynchronous distributed system[1].

Interaction-specific knowledge sharing: The good news is that only a small proportion of the pool of available knowledge will be of use to any given consumer, since each must have an upper limit on how much knowledge it can process. A pragmatic aim of open knowledge sharing, then, is to obtain knowledge *appropriate to the activities* in which each consumer wants to engage, while maintaining free and (adequately) reliable connections between suppliers and consumers.

The standard way in which activities (and their sequencing) are described is via process languages like BPEL [2] or LCC [14], since no complex activity can be represented formally without modeling its temporal structure. In principle, we could use (models of) these activities to limit the scope of knowledge that we attempt to share. There is a problem however: activity models are themselves knowledge that must be shared. In other words, when an item of knowledge is openly shared in the context of some common activity it is necessary for the supplier and consumer to have knowledge of that context, otherwise there is no benefit (in terms of reliable knowledge sharing) from the activity focus.

For this reason the OpenKnowledge project has at its core a mechanism for sharing models of activities that require interaction across the Internet. We refer to such models as *interaction models* [14]. We expect that communities of practice will naturally form around collections of interaction models and that these communities can be stabilized by a mechanism for their rapid sharing across peer groups. Notice that this is explicitly an *interaction-centered* approach to knowledge sharing, as opposed to the traditional *data-centered* approach.

By building a system, we demonstrate that sharing interaction models at very low cost to consumers and suppliers is possible. The novelty of this system is that each interchange of knowledge is made in the context of the (shared) interaction model. The system is completely distributed using P2P technology. Each peer that participates in the OK system will at least run a piece of code that we call the *OpenKnowledge Kernel* [4] enabling the base functionality to find these interactions and the code or peers that enable to run the services.

2 Relevant Literature

Clearly, many others have previously identified the goals of reliably sharing knowledge freely and reliably, regardless of the source or consumer. In this paper, we will not discuss the plethora of work in the dominant data-oriented attempts at solving this problem, such as data-integration [10], schema and ontology mapping [15], data-mediators [7], etc. Instead, in this section we discuss some of the approaches that have also taken an interaction-oriented approach: web-services, grid-services and multi-agent systems. Although typically data-centric, we also include P2P systems in our comparison, because the OpenKnowledge architecture has strong P2P characteristics

We do not aim to provide a full-scale literature study here. Instead, we identify the key ideas behind each of these approaches, and argue why OpenKnowledge occupies a unique niche in this landscape.

[1] Even if it were a philosophically and culturally coherent notion.

Web Services. Perhaps the most closely related effort to OpenKnowledge is the work on web-services [3]. The aim of web-services is to enable invoking and executing of services in a distributed, scalable and interoperable manner. The work on *semantic* web-services [19] adds to this the goals to automatically locate and compose such services in an open and heterogeneous environment like the Web.

Both approaches (web-services and OpenKnowledge) use the principle that if the services are formulated into information objects (web-service *descriptions* either purely syntactic, such as WSDL [5] or semantic such as WSDL-S [17] and OWL-S [11]), then they can also be the subject of reasoning tasks for search and composition.

The OpenKnowledge approach is in some ways more flexible than the web-services approach, but in other ways more restricted. Semantic web-service work aims at automatic on-line composition of simple services into complex services, by means of intelligent algorithms (e.g. based on configuration [20] or planning [21]), whereas, OpenKnowledge restricts itself to executing predefined "work-flows" of services (the "interaction models" to be discussed later in this paper). The only decision that OpenKnowledge makes at run-time is which instance of a service is executed; that is, which agent providing the service will be used (i.e. "recruiting", not composition).

This recruiting aspect of OpenKnowledge is more general than the web-service architecture because it separates the advertising of a service from the execution of a service. In the web-service architecture, it is generally assumed that advertisements of service functionality are accompanied with the name of the executor of the service. In short: the matching goals of both approaches are the same (finding a service that matches a given functionality), while the composition goals of both approaches are different: OpenKnowledge aims to recruit peers to execute predefined work-flows, whereas semantic web-services aims to automatically compose complex work-flows out of atomic services.

Furthermore, OpenKnowledge explicitly acknowledges the need for approximate matching of service requests with advertisements, whereas this is only marginally the case in the semantic web-service world [1], and entirely absent in regular web-services.

Finally, OpenKnowledge aims explicitly for a distributed storage model for the work-flows and service descriptions, whereas all the dominant web-service architectures (UDDI [12] for regular web-services, WSMX [8] for semantic web-services) assume a centralised architecture.

Grid-Services. The general area of grid-services is even less well circumscribed than web-services, hence it is more difficult to make a crisp comparison. Literature on Grids [6] often align their approaches to the service-oriented architecture (SOA). In contrast to web-services, grid-services are typically organized in fixed work-flows. This makes them more similar to the OpenKnowledge approach, however, grid-services emphasise various aspects that are ignored in OpenKnowledge: long-term stability of services, provenance, quality of service and resource monitoring. Similar to web-services, grid-services differ from Open Knowledge by advertising a service functionality together with the identification of the service-provider; OpenKnowledge decouples these two and hence allows for a separate "recruiting" step. Finally, and perhaps most importantly, most grid-systems provide only a centralized mechanism for advertising services and work-flows, while OpenKnowledge aims for a fully distributed mechanism.

In particular, the myGrid project [18] is in many respects close to the goals of OpenKnowledge in its use of pre-configured work-flows and its approach to manual composition of such work-flows. However, it relies on centralized storage of such work-flow patterns, which is in sharp contrast with the fully distributed architecture of OpenKnowledge.

Peer-to-peer systems. Obviously, OpenKnowledge is close in spirit to the work on peer-to-peer (P2P) systems. The central P2P ideas of distributed storage, lack of centralized address registers and the symmetric roles of every peer as both provider and requester, are fully adopted by OpenKnowledge. Nevertheless, OpenKnowledge makes two important deviations from most P2P systems. First, most P2P systems aim at data sharing, whereas OpenKnowledge aims at *service sharing*. Of course, data sharing is simply a special case of service sharing (namely sharing a data-access service), making the OpenKnowledge system more generic. Secondly, OpenKnowledge is in the small, but rapidly growing, family of *semantic* P2P systems [16], which use rich descriptions of the content that each peer has to offer for purposes of routing queries through the network.

Agents. A final class of closely related systems is that of multi-agent systems. In general, there is a superficial similarity between multi-agent and P2P systems: distributed sets of autonomous processes exchanging information. However, on closer inspection, there are rather significant differences. In particular, agent systems often have highly structured architectures inside each agent often relying on cognitive metaphors for their architectural constructs (such as the Believes, Desires and Intentions (BDI) architecture [13]). P2P systems typically treat their peers as atomic. Finally, agent-systems emphasize their pro-active nature (autonomously reacting on their changing environment), while P2P systems, including OpenKnowledge, assume more classical reactive stance.

The differences and similarities described above are all summarized in Table 1. This table shows that OpenKnowledge inherits many aspects from other approaches but also occupies a particular niche, having features not fully explored by others.

3 An Extensive Example Describing the Functionality of the OpenKnowledge System

In this section we provide an extensive example how our system can be used. The architecture of the system is described in another paper [4]. From a user perspective, the OpenKnowledge system is a software bundle that allows a user to find, compose and execute tasks. Those tasks can be executed by users and/or software components. The tasks are described by *Interaction Models* (IM), where each IM is a formally described set of roles together with the process-flow between those roles. Users subscribe their peer to play roles within an interaction. For example, the task of buying an item requires at least the seller and buyer roles, and perhaps a payment service role. We call instances of these roles (e.g. a particular seller or a particular buyer) *OK-Components (OKCs)*. An OKC, for example a creditcard service, may play a role in many IMs. If the roles are constrained by some external functionality, then services provide that functionality. Much of the functionality of the OK system relies on the *Discovery and Team formation Service (DTS)*, which is a distributed storage and retrieval system over a P2P network. Its main responsibilities being the following:

Web-Services	**similarities:**	service-oriented,	
		distributed,	
		automated search	
		based on semantic descriptions	
	differences:	**Web-Services**	**OpenKnowledge**
		composition of atomic services	predefined workflows
		fixed link to executing party	dynamic recruiting
		centralised advertising	distributed
		equivalence matching	approximate matching
Grid-Services	**similarities:**	service-oriented,	
		fixed workflows	
		distributed	
	differences:	**Grid-Services**	**OpenKnowledge**
		provenance	absent
		QoS	reputation mechanisms
		resource monitoring	absent
		centralised advertising	distributed
		fixed link to executing party	dynamic recruiting
Peer-to-Peer Systems	**similarities:**	distributed,	
		scalable,	
		symmetric roles of each peer	
	differences:	**P2P Systems**	**OpenKnowledge**
		aimed at data-sharing	service sharing
		independent of content	exploit semantics
Multi-Agent Systems	**similarities:**	distributed,	
		symmetric roles of each peer	
	differences:	**Multi-Agent Systems**	**OpenKnowledge**
		cognitive architecture	none
		central brokers	scalable discovery
		pro-active behaviour	reactive

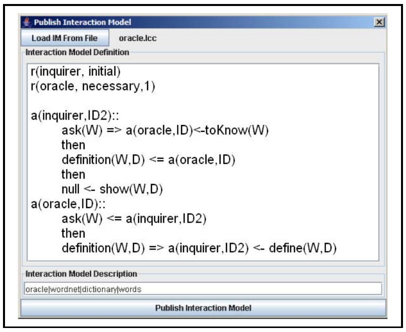
Fig. 1. OpenKnowledge compared to other approaches

- *IM Discovery* - the DTS is used to publish, discover and retrieve IMs.
- *OKC Discovery* - the DTS is also used to publish, discover and retrieve OKCs. This enables reusability thus providing scalable functionality. OKCs can be discovered either in the context of an already known IM or independently.
- *Role subscription* - peers can subscribe a locally stored OKC to play a role in an IM. Additional information such as annotations and restrictions concerning the other participants can be given along with the subscription.
- *Coordinator subscription* - peers may also subscribe to act as interaction coordinators.
- *Team formation and interaction initialization* - the DTS uses subscription information to form teams of OKCs, which will, potentially, participate in an interaction, and finds a subscribed coordinator to orchestrate them.

The system is based on previous work where the algorithms are simulated and implementations are emulated in order to see the performance of them [9]. More about the DTS can be read in the architecture paper [4]. Now we will explain the functionality of the first OpenKnowledge system by going through an example where we show how a dictionary service can be created and used.

3.1 Writing and Publishing an IM

In figure 2 user A uses the OpenKnowledge System to develop an IM for the dictionary service, by describing an interaction between two roles. One role is used to query the service, called the *inquirer*, and the *oracle* role provides the answer. In this example, the IM is written in the LCC language [14]. Current work in the project is to also have support to other languages like BPEL. The LCC model can be read as follows:


Publish Interaction Model
Load IM From File
oracle.lcc
Interaction Model Definition

```
r(inquirer, initial)
r(oracle, necessary,1)

a(inquirer,ID2)::
        ask(W) => a(oracle,ID)<-toKnow(W)
        then
        definition(W,D) <= a(oracle,ID)
        then
        null <- show(W,D)
a(oracle,ID)::
        ask(W) <= a(inquirer,ID2)
        then
        definition(W,D) => a(inquirer,ID2) <- define(W,D)
```

Interaction Model Description

oracle|wordnet|dictionary|words

Publish Interaction Model

Fig. 2. User interface showing an IM editor (LCC as the language in this example) and a button to publish the IM on the OpenKnowledge network

1. `r(inquirer,initial)`. This line states that the 'inquirer' role is the one that starts the interaction.
2. `r(oracle,necessary,1)`. Statement indicating that at least 1 peer needs to play the oracle role.
3. `a(inquirer,ID2)::`. A statement giving the 'inquirer' role an identifier 'ID2' and the '`::`' means that the definition of the role starts after it.
4. `ask(W) => a(oracle,ID) <- toknow(W)`. If the user wants to know a definition for a word 'W' it can start the interaction by fulfilling the constraint `toknow(W)`. In LCC the '`<-`' symbol is used to indicate that after it a constraint is defined. When the constraint is satisfied (i.e. the user provided 'W'), a message 'textttask(W)' is sent to the 'oracle' role identified by 'ID' (note that `a(oracle,ID)` relates the role to an identifier). In LCC the '`=>`' symbol is used to indicate that a message (in this case `ask(W)`) is sent from the current role to another role (in this case the 'oracle').
5. `definition(W,D) <= a(oracle,ID)`. In this line the 'inquirer' waits for the oracle role (`a(oracle,ID)`) to send a message with the definition as content (`definition (W,D)`). In LCC the '`<=`' symbol is used to indicate that a message (in this case `definition(W,D)`) should be expected from another role (in this case the 'oracle' role).
6. `null <- show(W,D)`. When the 'oracle' sent the message to this role, this statement shows the answer to the user. In this case `show` is a special constraint which is understood by the system to show a message (in this case with the query: `W` and the answer: `D`) in the user interface. `null` means that nothing happens after the constraint `show(W,D)` is fulfilled.
7. `a(oracle,ID)::`. Gives the 'oracle' role identifier 'ID' and starts to give its definition.
8. `ask(W) <= a(inquirer,ID2)`. This line makes the 'oracle' role wait for a message `ask(W)` from the 'inquirer'.
9. `definition(W,D) => a(inquirer,ID2) <- define(W,D)`. When the 'oracle' got the 'ask' message (previous line is executed), it will try to fulfill the 'define (W,D)' constraint, and if that is true, a message with the content `definition(W,D)` is sent to the 'inquirer'.

Now that a user `A` wrote down the IM, they should provide some keywords to describe the functionality of the IM. In the system we provide automated mapping and similarity algorithm to relate similar keywords during search.These keywords are needed by the DTS to index them in order to be retrieved by other peers. In this case, `A` decides to give the keywords '*oracle, wordnet, dictionary, words*'. Our current work tries to extend the ways to describe the functionality of an IM, for example by providing concepts from ontologies instead of keywords. Now that the IM is ready and the keywords are provided, the user can decide to publish it on the OpenKnowledge network by connecting to the network and pressing the 'Publish Interaction Model' button. The DTS will make sure it is scalably stored and indexed by the provided keywords.

3.2 Creating and Publishing OKC's

Besides writing the IM in the previous section, user `A` also writes the OKCs that implement both roles in the IM respectively. Currently, the user `A` has to implement their OKC by writing some code to a specific Java API. In simple terms, the methods in the Java source code should match the names and the arguments of the constraints in the roles, which are `toknow(W)` and `show(W,D)` for the 'inquirer' role and `define(W,D)` for the 'oracle' role. Note that here we assume `W` and `D` are of type `STRING`, where in the extended LCC language also types are supported, meaning that the definitions would be something like `show(W:STRING,D:STRING)`. After user `A` has implemented the interfaces, (s)he opens the window from the OpenKnowledge Kernel software where it can wrap the code into OKC's (the figure is not shown here due to space constraints).

The user loads its IM and attaches the java implementations of the role constraints via the user interface of the kernel. Also the OKCs may be described by a set of keywords, because they can be used as role implementations for other IMs and therefore need to be indexed so that they can be retrieved by the DTS. The intuition behind this is that an OKC implementing a credit-card payment service can be used in many IMs. Also these keywords can be used in the OKC selection process that allows a user to select their preferred OKCs after multiple matches have been found to an IM. For example, it can be that two OKCs exactly match the same 'oracle' role but one delivers results in English and the other in Spanish.

By clicking the '*Create OpenKnowledge Component*' button, the OKC is created and ready to be used. By sending a 'subscribe' message to the DTS (not shown in the figures), it tells the network that it is able to execute the role of 'oracle' for the given IM. Given that the user used Wordnet as the underlying implementation, it annotates the OKC with the keywords '*dictionary, english, wordnet,lookup*' (not shown in the figures). Besides this, `A` decides to publish the 'inquirer' OKC to the network, so that other users also may download it and run it on their own machines.

3.3 Searching for IMs and OKCs

Peer B wants to find a service that will allow it to find definitions of words in Spanish. It opens the search window from the OpenKnowledge Kernel (not shown due to space constraints). In this case, in the beginning (s)he searches for IMs matching to the word 'oracle'. The system starts searching and shows the found IMs together with their roles to the user. Assume that user `B` finds the IM together with the roles 'orcale' and

’inquirer’. The user wants to play the role of the inquirer written by user A and therefore decides to download it and tells the DTS that it is willing to play the role.

3.4 Team Formation and Execution

Given that in the previous steps A and B have both told the DTS that by subscribing their OKCs that they are willing to play the roles of ‘oracle’ and ‘inquirer‘ respectively, the DTS knows that all roles are instantiated meaning that there are enough peers to start the interaction. Now imagine that another user C also published an OKC that is able to fulfill the role of ‘oracle’, but has annotated its OKC with the keywords *dictionary, spanish*. So now there are three peers ready to play. The DTS selects a coordinator peer from the pool of peers. This is currently selected randomly (but current ongoing work is to make it reputation-based). This coordinator receives a message from the DTS with the three peers, their OKC descriptors and the IM. The coordinator now can start the team formation process.

The coordinator sends each peer the list of peers willing to play together with their OKC descriptions. Now the peers can select, automatically or with the user in the loop (depends on the OKC implementation), with whom to play. Assume that both the Spanish and English oracles have automatic selection process saying that they always like to play with whomever. However, the inquirer has user B in the loop, where the user selects the peer from user C, because its OKC description matches its wishes and sends its preferences back to the coordinating peer. Now that the coordinator has (within a certain time-out) received enough replies to start the interaction, its starts executing it. The coordinator sends a message to Peer B which solves the constraint by asking the user (using a visualizer showing the constraint to the user). The word is sent back to the coordinator which continues parsing the IM and reaches a constraint that must be satisfied by the dictionary role to give the word definition. The coordinator sends the constraint to Peer C which solved it and returns the definition in a message. The coordinator continues parsing and finds a constraint in which the querier role must show the user the word definition. It sends Peer B a message with this constraint and it is solved by showing the query results to the user. The IM is finished at this point, so the coordinator sends a message to each peer so they can stop the OKC instances.

As said, this example demonstrates the functionality of the system, but it is very simple. The interface presented is only one of the many possible interfaces, because we have designed the architecture to be as independent as possible from the user presentation system.

3.5 Other Examples

Some interesting examples can be made within the trade domain, like an interaction model for a transaction of goods. Somebody may publish an IM that contains the process-flow between a seller, a buyer and a payment service. Peers can subscribe themselves to these roles and when all roles are instantiated the interaction starts. The *Coordinator* initiates the interaction and coordinates it. Especially in this case, all role-players may want to have a trustworthy controller, and can specify the requirements for a coordinator when subscribing to an OKC.

Another example comes from a case study that we undertook in the bio-informatics domain [22]. In that paper we present a system that can be used to analyse real data of relevance to the structural bio-informatics community where comparative models of yeast protein structures from different resources are analysed for consistency between them. The interaction model described in that paper, written in the LCC language, describes the interaction between the roles of data collector, receiver and source, that together perform the task.

4 Summary

Much of the information that might be accessed in semantic webs is accessible through complex programs (web-services, sensors, *etc.*) that may interact in sophisticated ways. Composition guided simply by specifications of programs' input-output behaviours is insufficient to obtain reliable aggregate performance - hence the recognised need for process models to specify the interactions required between programs. These interaction models, however, are traditionally viewed as a consequence of service composition rather than as the focal point for facilitating composition. We have described an operational system that uses models of interaction as the focus for knowledge exchange. Our implementation adopts a peer to peer architecture, thus making minimal assumptions about centralisation of knowledge sources of interaction control. The direct contribution of this paper is to present the first operational system of this kind. The secondary contribution of this paper is to provide a new angle on service orchestration and ontology matching that re-interprets traditional methods for these tasks in a dynamic context.

Acknowledgements. This work has been supported by the FP6 OpenKnowledge project[2]. A. Perreau de Pinninck is supported by a CSIC predoctoral fellowship under the I3P program, which is partially funded by the European Social Fund.

References

1. Akahani, J., Hiramatsu, K., Kogure, K.: Coordinating Heterogeneous Information Services based On Approximate Ontology Translation. In: AA MAS 2002. First International Joint Conference on Autonomous Agents & Multiagent Systems (2002)
2. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services, version 1.0. Technical report (2004)
3. Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing 6(2), 86–93 (2002)
4. de Pinninck, A.P., Dupplaw, D., Kotoulas, S., Siebes, R.: The openknowledge kernel. In: Proceedings of the IX CESSE conference, Vienna, Austria (2007)
5. Meredith, G., Weerawarana, S., Christensen, E., Curbera, F.: Web services description language (wsdl) 1.1. Technical report (2001)
6. Foster, I., Kesselman, C.: The grid: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
7. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V., Widom, J.: The TSIMMIS Approach to Mediation: Data Models and Languages. Journal of Intelligent Information Systems 8(2), 117–132 (1997)

[2] http://www.openk.org/

8. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX-a semantic service-oriented architecture. In: Proceedings IEEE International Conference on Web Services, 2005. ICWS 2005, pp. 321–328. IEEE Computer Society Press, Los Alamitos (2005)
9. Kotoulas, S., Siebes, R.: Adaptive routing in structured peer-to-peer overlays. In: 3rd Intl. IEEE workshop on Collaborative Service-oriented P2P Information Systems (COPS workshop at WETICE07), Paris, France, IEEE Computer Society Press, Los Alamitos (2007)
10. Lenzerini, M.: Data integration: a theoretical perspective. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 233–246. ACM Press, New York (2002)
11. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al.: OWL-S: Semantic Markup for Web Services. W3C Member Submission 22 (2004)
12. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Importing the Semantic Web in UDDI. Web Services, E-Business and Semantic Web Workshop (2002)
13. Rao, A.S., Georgeff, M.P.: Modeling rational agents with a BDI-architecture. Readings in agent, 317–328 (1997)
14. Robertson, D.: A lightweight coordination calculus for agent systems. In: Leite, J.A., Omicini, A., Torroni, P., Yolum, p. (eds.) DALT 2004. LNCS (LNAI), vol. 3476, pp. 183–197. Springer, Heidelberg (2005)
15. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. Journal on Data Semantics IV, 146–171 (2005)
16. Siebes, R., Kotoulas, S.: proute: Peer selection using shared term similarity matrices. Web Intelligence and Agent Systems 5(1), 89–107 (2007)
17. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J.: Adding Semantics to Web Services Standards. In: Proceedings of the International Conference on Web Services, pp. 395–401 (2003)
18. Stevens, R., Robinson, A., Goble, C.A.: mygrid: Personalised bioinformatics on the information grid. In: proceedings of 11th International Conference on Intelligent Systems in Molecular Biology, Brisbane, Australia (2003)
19. Sycara, K.P., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. J. Web Sem. 1(1), 27–46 (2003)
20. ten Teije, A., van Harmelen, F., Wielinga, B.: Configuration of web services as parametric design
21. Traverso, P., Pistore, M.: Automated Composition of Semantic Web Services into Executable Processes. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, Springer, Heidelberg (2004)
22. Gerloff, D., Sharman, J., Quang, X., Walton, C., Robertson, D.: Peer to Peer Experimentation in Protein Structure Prediction: an Architecture, Experiment and Initial Results. In: International Workshop on Distributed, High-performance and Grid Computing in Computational Biology, Eilat, Israel (2007)

Ontology Enrichment in Multi Agent Systems Through Semantic Negotiation

Salvatore Garruzzo and Domenico Rosaci

DIMET, Università Mediterranea di Reggio Calabria
Via Graziella, Località Feo di Vito
89122 Reggio Calabria, Italy
{salvatore.garruzzo,domenico.rosaci}@unirc.it

Abstract. Ontologies play a key role in the development of Multi-Agent Systems (MASs) for the Semantic Web, providing conceptual description of the agents' world. However, especially in open MASs, agents use different ontologies and this often leads to communication failures. Semantic negotiation is a recent framework which provides an effective solution to such a problem, but it is a too heavy framework to be implemented in large agent communities. In this paper, we deal with the inefficiency in semantic negotiations, and we show how a possible solution is to build a common representation of the different terms used by the agents. We argue that a reasonable compromise to use a common ontology consists of combining it with semantic negotiation and we propose an algorithm which implements this idea in the recent HISENE semantic negotiation framework. Moreover, the semantic negotiation is exploited, in our proposal, to dynamically enrich the global ontology.

1 Introduction

Nowadays, we can observe a growing use of MASs in different applications on the Semantic Web, since software information agents make possible the widespread acquisition of machine understandable data, opening myriad opportunities for automated information processing. In this context, the notion of *ontology* plays a prominent role. On one hand, ontologies drastically enhance the possibility to make the Web being really "semantic". On the other hand, in order to make both effective and efficient the use of ontologies, two main problems, strongly related to the intrinsical heterogeneity of the Semantic Web, arise.

Problem A: Heterogeneity between agents. Ontologies are often advocated as a complete solution for knowledge sharing between agents, giving the possibility to assign a *meaning* to terms contained in the exchanged messages. However, such a possibility exists only in the case each agent of the system knows the ontology of other agents; on the contrary, an agent that receives a message from another one that uses a different ontology is not able to understand the content of the message. A solution to such a problem is represented by the use of a common ontology [4,6], shared by all the agents. However, this is a solution that appears unlikely in open MASs, since it would imply all the agents agree to adopt a standard ontology, about which it is necessary to reach consensus.

Problem B: Necessity of a unique representative ontology. An agent has the necessity to know the content of the other ontologies, in order to choose the most suitable terms for a correct communication. In other words, the agent would desire to have a "global ontology" which allows him to interact with the community. Moreover, such a global ontology would be also very useful for the new agents that join with the community. However, for the same reason, it is worth to point out that the global ontology cannot be "a priori" fixed and static, since it must reflect the possible introduction of new terms and the possibility of use of new meanings of the same term.

Recently, a new framework suitable to face the problem of semantic heterogeneity has been developed, which seems promising to solve the problem A. This framework, called *semantic negotiation* [3,5,6], is a process by which the agents of a MAS try to reach mutually acceptable definitions (i.e., mutually acceptable agreements on terms). In this context, in [1] we have introduced the idea that two agents involved in a communication can require the help of other agents in order to solve possible understanding problems. On the basis of this idea, we have proposed the *HIerarchical SEmantic NEgotiation* (HISENE), that is suitable to be applied for implementing such a semantic negotiation in the standard Java Agent DEvelopment Framework (JADE) [2]. HISENE gives a solution to the problem A since its semantic negotiation protocol provides a framework to allow the agents of a MAS to understand each other, without constraining the agents to adopt a unique, fixed ontology. However, if a new agent joins with the system, he cannot access to a global view of the existing terms, but he needs to activate a semantic negotiation to gradually learn the personal "language" of each other agent. This obviously leads to significant inefficiencies in the communication process of the entire system. The present work gives a contribution to the problem B. In particular, we propose an algorithm, called *Hisene Ontology Enrichment* (HOE), to derive a global ontology from the personal ontologies of different agents. The global ontology generated by our algorithm contains all the terms used by each agent and, for each term, the set of all the different meanings exploited for that term. Moreover, the global ontology so derived can be continuously enriched during the evolution of the system, giving the possibility to add new terms and new meanings of the same term. Using HOE in combination with HISENE, each agent of a MAS can autonomously enrich his own ontology by using the semantic negotiation protocol and, at the same time, access to the global ontology to have a synoptical view of the terms used by all the other agents. Each term of the global ontology is associated with a set of meanings, and each meaning is associated, in its turn, with the agents that have used it in the past. This allows an agent, that desires to send a message to another agent, to choose the most suitable term with the most appropriate meaning from the global ontology. Only in the case the agent does not find in the global ontology the necessary term, he will use a new, personal term that is not contained in the global ontology and that probably will lead to a semantic negotiation process.

This way, the use of the semantic negotiation, which is a significantly onerous task, is limited to the strictly necessary cases. The plan of the paper is as follows: Section 2 deals in detail with the HISENE protocol, while Section 3 describes the HOE algorithm. Finally, Section 4 draws some conclusions.

2 HISENE2: A Protocol to Support Semantic Negotiation

In this section, we briefly describe the protocol HISENE2, expressly conceived to support semantic negotiation between agents that exploit different ontologies in an open MAS. HISENE2 has been implemented under the JADE framework, that is an agent development environment fully compliant with FIPA specification. In JADE, an ontology is a set of schemas defining the structure of the concepts that are pertinent to the domain of interest. We refer below to the JADE ontology in order to define our *Extended Ontology*, which is formed by a set of *elements*. Each element can be either a traditional JADE ontology schema, that here we simply call *concept*, or an *explained element*. More in detail, an explained element is a set of *explanations*, where the new notion of *explanation* can be considered as a description of an element based on other elements. In other words, we introduce a different way of representing the reality of an agent with respect to a classical JADE ontology. In fact, a JADE ontology contains only concepts (classes, object schemas), where each concept has a unique meaning expressed by both its class name (the lexical component) and its class structure (the structural component) and thus does not need any further explanation. Differently, an extended ontology contains also explained elements, that can have several meanings. Moreover, we consider that in the ontology of an agent i, each explanation is associated with a set of agents, that we call *context*, for which this explanation is understandable. Furthermore, we associate with each explanation the *explainer* agent that has provided it to i and the *confidence* that i assigns to the explanation. Communications between agents in JADE are held by means of messages having a format specified by the ACL language, defined by the FIPA international standard. In our framework, agents performing semantic negotiation activities need to *(i)* express the content of the message using the extended ontology, and *(ii)* exchange more other information (e.g. the list of the unknown elements). In order to satisfy the issue *(i)*, we introduce the concept of *semantic ordinary message* that extends the message format described below, by using only explained elements in its content. Moreover, to satisfy also the issue *(ii)*, we define the *semantic negotiation message* that further extends the semantic ordinary message.

Definition 1 (Semantic negotiation message). A *semantic negotiation message* is a tuple $\langle i, j, ia, p, T, C \rangle$ where i, j are the sender and receiver agents, respectively, ia is the agent interested in the understanding process, p is the performative, T is the understanding timeout determined by ia, and C is the content of the message.

Our protocol is composed by six performatives, namely:

1. `SN_QUERY` : the agent i requires the help of the agent j to understand some unknown elements. For this purpose i specify in the content C a list *ununderstood* of explained elements $(el_1, el_2, \ldots)$.
2. `SN_RESPONSE` : after receiving a `SN_QUERY` message from the agent j, the agent i replies giving some explanations. In this performative, the content C is a list of explained elements containing their explanations.
3. `SN_ACCEPT` : after receiving a `SN_RESPONSE` message from the agent j, the agent i indicates the understood explanations. In this performative, the content C is a list of explained elements containing the accepted explanations.
4. `SN_UNKNOWN` : the agent i is unable to give an answer to a previous `SN_QUERY` message sent by the agent j. The message's content is void.
5. `SN_ALREADY_ANSWERED` : as previously described in Section 1, an agent receiving a `SN_QUERY` message can start, in its turn, another semantic negotiation; as a consequence, an agent can receive the same `SN_QUERY` message from different agents. In this scenario, after receiving a `SN_QUERY` message from the agent j, the agent i replies that it has already answered to the same request previously received. The message's content is void.
6. `SN_FEEDBACK` : the agent i is unable to understand a semantic ordinary message even after a semantic negotiation. In this scenario, i replies indicating the not understood explanations specified in the content C.

An agent supporting the semantic negotiation can perform three different behaviours, namely:

A **request behaviour** is started when an agent i needs to understand unknown explained elements. This happens when i receives: (i) a semantic ordinary message that i does not understand. The agent i creates the message $\langle i, j, i, SN\_QUERY, T, C\rangle$, where j is the generic receiver agent, i itself is the interested agent and C is the list *ununderstood* of unknown explained elements; (ii) a `SN_QUERY` message having in its content C some explained elements that i is unable to explain. Thus, i creates the message $\langle i, j, ia, SN\_QUERY, T, C^*\rangle$ where j is the generic receiver agent, the interested agent is the same of the received message, and $C^* \subseteq C$ is the list *ununderstood.* A function `createPartitions` reads the expertise and reputation coefficients and, on the basis of the partition weights set by the agent owner, determines the agent partitions. Then, `SRequest` and `SReceive` behaviours are executed. `SRequest` is a OneShotBehaviour that, for each partition level k, sends a `SN_QUERY` message to each agent contained in the k-th partition, until either the list *ununderstood* becomes empty or the message timeout is reached. `SReceive` is a CyclicBehaviour in which i waits for answers from the contacted agents. Each received `SN_RESPONSE` message has as content a list of explained elements $(el_1, el_2, \ldots, el_h)$ where el_m contains the explanations $\{e^1_m, e^2_m, \ldots, e^l_m\}$. Therefore, the function `solveSemanticUnunderstanding` `(`e^g_m`)`, $g = 1, 2, \ldots, l$ is called for each received explanation e^g_m. This function performs a schema matching between the i's ontology and the set of elements contained in e^g_m. For each accepted explanation e relative to an explanation element el, the agent i (i) stores e inside the element el in its ontology and *(ii)* replies a `SN_ACCEPT` message indicating the understood explanation.

An **answer behaviour** is started when an agent i receives a request message (`SN_QUERY`). There are three possibilities: (i) i has previously answered to the same message. In this case, i replies with a `SN_ALREADY_ANSWERED` message; (ii) i understands the message. In this case i replies with a `SN_RESPONSE` message containing the list of explained elements as described above; (iii) i does not understand the whole message. In this case, i starts in its turn a request behaviour (i.e. a new semantic negotiation). After that, if the message is partially or completely understood (resp. not understood), i replies with a `SN_RESPONSE` (resp. `SN_UNKNOWN`) message.

A **feedback behaviour** is started when an agent k receives a semantic ordinary message containing some unknown explained elements from an agent i. The agent k begins a semantic negotiation in order to understand all the unknown explanations. After having concluded this negotiation, k sends to i a `SN_FEEDBACK` message containing the list of all the explanations that remained already unknown. The agent i, for each explanation e learnt from the agent j that is contained in a `SN_FEEDBACK` message updates the explanation confidence. and both the reputation and expertise coefficients of j.

3 The HISENE Ontology Enrichment (HOE) Algorithm

In this section we describe the HOE algorithm, which exploits the result of the semantic negotiation to construct and incrementally enrich the global agent ontology of the whole system. To informally describe the idea underlying HOE, we propose an example of how this algorithm works. Consider the simple MAS graphically depicted in Figure 1, composed by the three agents A, B and C. Each agent has its own ontology, which is continuously enriched by performing semantic negotiation activities: we have denoted by O_A, O_B and O_C the ontologies associated with A, B and C, respectively. In particular, Figure 1-(a) shows an initial situation, in which the agents A, B and C have only unexplained elements in their associated ontologies, since no semantic negotiation process has yet been performed. In such an initial situation, as shown by the Figure 1-(a), the global ontology of the MAS, denoted by O_{MAS} is simply the union of the three personal agent ontologies O_A, O_B and O_C. Now, as a first situation, suppose that C sends to B a query message, asking if B has a *plant* of Rome (see Figure 1-(b)). Unfortunately, B does not have in his ontology the element *plant*. As a consequence, we suppose that B begins a semantic negotiation with C, by asking him to explain what is a *plant*. We assume that C is able to explain *plant* by using another element of his ontology, namely *map*. This way B understands the meaning of *plant* since he has the element *map* in his own ontology, and he is able to answer C. Consequently, both B and C now adds to their ontologies the new explanation of *plant* in term of *map*, by also recording that B is the explainer. We also suppose that both B and C will use in the future the new explanation of *plant*, and since they will presumably obtain good results in most of the cases, the confidence of the explanation will remain equal to 1. Consider now a second situation. Suppose that C sends to A a message which contains

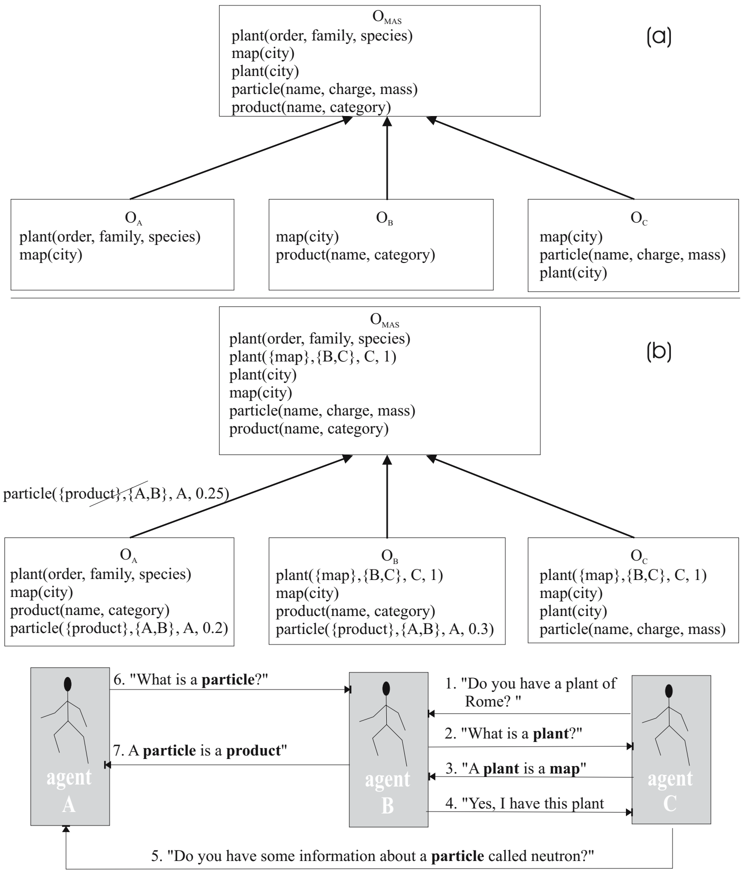

O_MAS
plant(order, family, species)
map(city)
plant(city)
particle(name, charge, mass)
product(name, category)
(a)
O_A
plant(order, family, species)
map(city)
O_B
map(city)
product(name, category)
O_C
map(city)
particle(name, charge, mass)
plant(city)
O_MAS
plant(order, family, species)
plant({map},{B,C}, C, 1)
plant(city)
map(city)
particle(name, charge, mass)
product(name, category)
(b)
particle({product},{A,B}, A, 0.25)
O_A
plant(order, family, species)
map(city)
product(name, category)
particle({product},{A,B}, A, 0.2)
O_B
plant({map},{B,C}, C, 1)
map(city)
product(name, category)
particle({product},{A,B}, A, 0.3)
O_C
plant({map},{B,C}, C, 1)
map(city)
plant(city)
particle(name, charge, mass)
6. "What is a particle?"
7. A particle is a product"
1. "Do you have a plant of Rome? "
2. "What is a plant?"
3. "A plant is a map"
4. "Yes, I have this plant
5. "Do you have some information about a particle called neutron?"
agent A
agent B
agent C

Fig. 1. The state of the ontologies (a) before and (b) after the semantic negotiation

the element *particle*. This element is not present in the ontology of A, that tries to understand it by performing a semantic negotiation task. Suppose that B, that is involved in the semantic negotiation, gives an explanation of *particle* in terms of another element of his ontology, i.e. *product*. Obviously, this is a wrong explanation, probably due to a misunderstanding of *particle* (for example, the human owner of the agent B might have confused *particle* with the element *article* and consequently associated it with *product*). The new explanation of *particle* in terms of *product* is then added in the ontology of both A and B, recording B as explainer. Now, we can easily suppose that in the future A and B will use this explanation of *particle*, obviously having bad results. Consequently, we argue that the confidence in this explanation will rapidly decrease in time (in Figure 1-(b) we have supposed the values 0.2 and 0.3 in the ontologies of A and B, respectively). The two situations described above are two examples of how the semantic negotiation leads to an enrichment to the personal ontologies of the agents. As a natural extension of the approach described above, we now propose to add the new explanations, derived by semantic negotiation, in the global ontology of the MAS, such that the global ontology is always the union of the elements present in the personal ontology. However, we point out that it is not suitable to add in the global ontology those explanations which have a small confidence, since they should be considered as bad explanations. In order to evaluate what value of confidence to assign to an explanation that is present at the same time in different ontologies, we associate to it, as confidence coefficient, the average of the corresponding confidence coefficients in the personal ontologies. For instance, the confidence coefficient associated to *particle({product},{A,B}, A, 0.25)*, is computed as the average of the confidence 0.2 relative to the ontology of the agent A and the confidence 0.3 relative to the ontology of the agent B, and thus it is equal to 0.25. In the example of Figure 1, we have decided to add to the global ontology only those explanations which have a confidence greater than or equal to 0.5, therefore the explanation *particle({product},{A,B}, A, 0.25)* is discarded, as graphically represented in Figure 1-(b). The enrichment of the global ontology allows to introduce a significant improvements in the semantic negotiation protocol. Indeed, in the new version of the protocol that we here propose, when an agent x desires to send a message to another agent y, first he examines the global ontology to find possible explanations of elements that contains y in their context. This avoids the use of elements that y cannot understand and will reduce the use of semantic negotiation. On the other hand, when an agent x receives a message from another agent y, and that message contains some elements which do not belong to the ontology of x, then x examines the global ontology for finding possible explanations of these elements, that he is able to understand. Only in the case the exam of the global ontology does not success, x performs a semantic negotiation task. Now we describe the behaviour of an agent a which receives a message and uses the HOE approach. If the agent does not understand the message, then he firstly calls the function `GlobalOntologySearch`. This function receives as input the ontology O_a of the agent a, the global ontology O_{MAS} of the MAS and the set U which contains all the elements of the message m which a does not understand. The function, for each unundestood element u which belongs to U, checks if u belongs to the global ontology O_{MAS}. Here we assume that u belongs to O_{MAS} if O_{MAS} contains an element named u, without considering the content of u. In the positive case, it examines each explanation e of the element u contained in O_{MAS}. We remember that the explanation e is a tuple $\langle E, C, ea, c\rangle$ where E is the set of ontology elements constituting e, C is the context, i.e. the set of the agents which are able to understand e_u, ea is the explainer agent that provided e, and $c \in [0, 1]$ is the explanation confidence. If the set E is also contained in the ontology O_a of the agent a, then the agent a understands the explanation e and consequently e is added, by using the function `add`, to the element u in the ontology O_a. After the execution of `GlobalOntologySearch`, if the message has been understood, the *HOE* behaviour ends, without performing any semantic negotiation. Otherwise, if some elements e_1, e_2,..,e_k are yet unundestood, then the answer behaviour is executed relatively to only these elements. When this behaviour is terminated, the function `GlobalOntologyUpdate` is called. This function receives as input the set U which contains the elements which has been involved in the previous Answer behaviour, and the global ontology O_{MAS}. If the generic element u is not present in O_{MAS}, the function simply adds u to O_{MAS}, by using the function `addElement`. Otherwise, if u is already present in O_{MAS}, the function updates the set of the explanations of u in O_{MAS}. The update is performed as follows. If the element u belonging to U contains an explanation e which is not also contained in the corresponding element of O_{MAS}, and the confidence of e, say c_e, is greater than the threshold σ, then this explanation is added to O_{MAS}. Otherwise, if the element u belonging to U contains an explanation which is also already contained in the corresponding element of O_{MAS}, then the confidence coefficient of the explanation in O_{MAS} is updated. In order to describe how the update of the confidence coefficient is done, let c_e^U be the confidence associated with the explanation e in the list U, and let c_e^{MAS} be the confidence associated with the same explanation in the global ontology O_{MAS}. The new value of c_e^{MAS} is computed as the average between the old value of c_e^{MAS} and the value c_U.

4 Conclusions

In conclusion, we have highlighted that the main reason of inefficiency in semantic negotiation is that the knowledge acquired through the negotiation activities is not suitably shared among the agents. To overcame this limitation, we propose to construct a global ontology of the MAS, that can be dynamically enriched exploiting the results of the semantic negotiation. The HOE algorithm for the global ontology enrichment introduces a relatively little cost for updating the ontology elements, being its execution distributed on the whole community. On the other hand, the combined use of the semantic negotiation and the global ontology generates a drastically reduction of the communication cost.

References

1. Garruzzo, S., Rosaci, D.: HISENE2: A Reputation-based Protocol for Supporting Semantic Negotiation. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 949–966. Springer, Heidelberg (2006)
2. http://jade.tilab.com (2005)
3. Soh, L., Chen, C.: Balancing ontological and operational factors in refining multiagent neighborhoods. In: Kudenko, D., Kazakov, D., Alonso, E. (eds.) AAMAS 2005. LNCS (LNAI), vol. 3394, pp. 745–752. Springer, Heidelberg (2005)
4. van Diggelen, J., Beun, R.-J., Dignum, F., van Eijk, R.M., Meyer, J.-J.Ch.: Optimal communication vocabularies and heterogeneous ontologies. In: van Eijk, R.M., Huget, M.-P., Dignum, F.P.M. (eds.) AC 2004. LNCS (LNAI), vol. 3396, Springer, Heidelberg (2005)
5. van Diggelen, J., Beun, R.-J., Dignum, F., van Eijk, R.M., Meyer, J.-J.C.: An effective minimal ontology negotiation environment. In: AAMAS Int. Conf. ACM (2006)
6. Williams, A.B.: Learning to Share Meaning in a Multi-Agent System. Autonomous Agents and Multi-Agent Systems 8(2), 165–193 (2004)

A Relaxed But Not Necessarily Constrained Way from the Top to the Sky

Katja Hose[1], Christian Lemke[1], Kai-Uwe Sattler[1], and Daniel Zinn[2]

[1] Dept. of Computer Science and Automation, TU Ilmenau
[2] Dept. of Computer Science, University of California, Davis

Abstract. As P2P systems are a very popular approach to connect a possibly large number of peers, efficient query processing plays an important role. Appropriate strategies have to take the characteristics of these systems into account. Due to the possibly large number of peers, extensive flooding is not possible. The application of routing indexes is a commonly used technique to avoid flooding. Promising techniques to further reduce execution costs are query operators such as top-N and skyline, constraints, and the relaxation of exactness and/or completeness. In this paper, we propose strategies that take all these aspects into account. The choice is left to the user if and to what extent he is willing to relax exactness or apply constraints. We provide a thorough evaluation that uses two types of distributed data summaries as examples for routing indexes.

1 Introduction

One of today's challenges in data integration and distributed data management is to cope with large-scale dynamic environments. A promising solution are Peer Data Management Systems (PDMS), which combine the peer-to-peer (P2P) paradigm and its characteristics such as self-organization, robustness, scalability, and the absence of global knowledge with ideas from classical federated databases. In a PDMS, each peer provides its own data with its own schema and in this way preserves the sovereignty over its data. Furthermore, each peer can answer and process queries and is linked to a small set of neighbors via mappings representing schema correspondences.

However, an inherent problem of large-scale dynamic data management systems is to guarantee complete and exact query answers. In principle, this requires querying all the peers in the system (e.g., by exhaustive flooding) or to know all peers holding relevant data (i.e., to have global knowledge). This is mostly impossible simply due to the mere size of the system. A possible improvement to avoid expensive flooding is the usage of routing indexes [2]. In their original sense they index files by means of keywords. A slightly different understanding of routing indexes came up later [9, 5]. These indexes use summarizing data structures to describe numerical attributes of data records. In accordance to [5] we call such routing indexes *Distributed Data Summaries* (*DDS*). Figure 1 illustrates the two variants (QTree-based [5, 11], histogram-based) we are using in this paper.

Because of the heterogeneity of the data as well as the autonomy and dynamicity expecting exact or complete results often makes no sense: mappings are incomplete, peers can join or leave the system at any time, data is dirty, etc. Thus, we argue that *relaxing* exactness/completeness expectations is a key for efficient query processing in PDMS. Such relaxations can be achieved on more than just one level. First of all, *ranking query operators* such as skyline and top-N by definition do not aim at providing a complete and detailed answer to a query. They provide an overview over the data in the system. Skyline queries [1] are the logical consequence of top-N queries, where the user is not only allowed to define one single ranking function but an arbitrary number of them. These user-defined ranking functions may differ substantially from one query to the next, so that any kind of preprocessed results would not help. Sometimes, however, the user is not interested in the whole data space but only in a subspace that he can specify with *constraints*. This leads to constrained skylines [3] and constrained top-N queries that only consider records in a subspace of the whole data space. Still, there is another option to reduce execution costs: relaxing the completeness/exactness requirements by allowing *fuzziness*. A representative of a fuzzy area may represent several result records clustered in one region.

In this paper, we build upon previous work [11,5,6,4] that discusses the use of DDS and relaxation for processing skyline queries. We present a strategy that adopts and enhances these techniques such that not only skyline but also top-N queries benefit from the application of a fuzzy parameter. We additionally introduce constraints and examine the benefits that we gain from two different types of DDS (QTree-based, histogram-based). The remainder of this paper is structured as follows. After having sketched related work in Section 2, Section 3 presents a strategy for processing relaxed queries in PDMS. In Section 4 we extend this strategy to constraints. Section 5 shows the results of our evaluation and Section 6 concludes this paper.

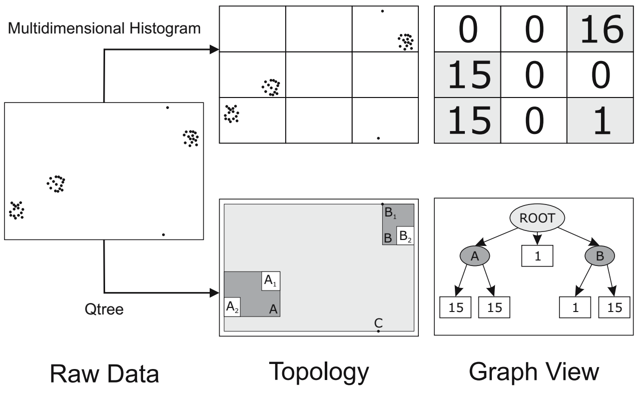


Fig. 1. Base Structures for DDS Illustrated at the Example of a Two-Dimensional Data Set. Left: Original Data, Center: Topology View with Regions/Buckets Representing the Data, Right: Graph View (QTree: Inner Nodes Depicted as Ellipses, Statistic Nodes as Rectangles; Histogram: Regions with the Number of Represented Records Corresponding to Statistic Nodes in the QTree)

2 Related Work

In contrast to the skyline operator, which came up in database research only a few years ago [1], the top-N operator had already played an important role in RDBMS before.

Later on, when distributed systems gained more importance, the Threshold Algorithm (TA) [8] and algorithms based on it have been developed. However, all approaches based on TA have some severe drawbacks: first, the vertical distribution of data and the specialized network structure, where the processing node has direct access to all other nodes. Second, TA requires sorted lists of all objects that we cannot assume to exist in P2P environments.

Already before [1] introduced the skyline operator into database research, the problem had been known as the maximum vector problem [7] before. However, most of the works published have been designed for centralized systems and only a few approaches consider computing skylines in distributed environments. Some exploit the TA principle of sorted lists for processing skylines. The same reasons as stated above make them hardly applicable to PDMS. Another recent work that considers skyline processing in distributed environments is DSL [10]. Although this approach works well for structured overlays, we focus on a more general solution that is able to process skylines in unstructured P2P networks, where we cannot influence the data a peer holds.

3 Distributed Query Processing

In this section we present our algorithm for processing relaxed rank-aware queries in a completely decentralized fashion using the information provided by DDS. Let us first give a formal definition of relaxation:

Given a set D of data objects, a top-N or skyline query $\mathfrak{T}$, a distance function $d : D \times D \to \mathbb{R}$, and a limit $\varepsilon \in \mathbb{R}$, then any subset R of D for that

$$\forall t \in \mathfrak{T}(D)\ \exists r \in R : \ d(t, r) \leq \varepsilon \tag{1}$$

holds, is called a *relaxed top-N/skyline result*. Furthermore, if for $r,\ t \ \in \ D$: $d(r, t) \leq \varepsilon$ holds, r is called a *representing record* of t. A *representative* is the combination of such a representing record and the region that is represented. Thus, a relaxed top-N/skyline result can be defined as a set R that contains a representative for each result record $t \in \mathfrak{T}(D)$. Note that there are usually many R for a data set D that fulfill Equation 1. Furthermore, several records can be represented by one single representative.

The guarantee that is output to the user for such a result is an inherent part of any representative. It guarantees that all records that are represented by the representative are situated within the region that is part of its definition. The maximum distance between the representing record and any point in the region never exceeds ε with respect to distance function d.

Algorithm. Since the ε value limits the maximum approximation error that the algorithm is allowed to make,ε and d have to be specified by the user and added to the query definition. The algorithm for relaxed rank-aware queries can be summarized by the following steps that need to be executed at each peer that receives the query:

1. Compute the query locally – considering local data, DDS regions, constraints, and if provided data received along with the query.
2. Try to find representatives for all regions that are part of step 1's result – using distance function d and the maximum distance ε received along with the query.
3. Forward the query to all neighbors whose regions could not be represented.

4. After all queried neighbors have answered: determine the result over the union of their answer data and the local result – both may include representatives.
5. Try to minimize the number of representatives.
6. Forward the result to the query's sender - data records as well as the remaining representatives.

In the following, we at first describe how to determine representatives for regions. Next, we discuss how to compute rank-aware queries over representatives and finally we discuss how to minimize the number of representatives. Due to limited space we focus on processing top-N queries.[1] All techniques including the application of constraints presented in Section 4 can be used for processing skyline queries as well.

Determining Representatives. Finding representatives for the data that is relevant to the query and provided by neighboring peers reduces execution costs. In the best case all regions that are part of the query result over regions (resulting from step 1) can be represented by known data records such that the query does not have to be forwarded at all. More precisely, choosing representatives means: Given a region B, a distance function d, and a distance ε we have to find a set of records D_{loc} that represents B with respect to the given query specification such that for any point $p \in B$ there exists a local data record $l \in D_{loc}$ for that $d(p,l) \leq \varepsilon$ holds. The problem we encounter is to find a minimal set of representatives that correctly represents a region. As a simple solution to this problem our implementation represents regions only if it is possible to represent them with one single representative, i.e., a region is represented if it is completely enclosed in the region defined by d, ε, and the representing data record.

Top-N Computation over DDS Regions. Without loss of generality let us assume that the score value, which is assigned to a data record by the ranking function, has to be minimized. Let $s_{max}(B)$ and $s_{min}(B)$ denote the maximum and minimum scores that any point in region B might have. Furthermore, let $count(B)$ denote the number of data records contained in B. Finally, let $\mathbf{B_{all}}$ be the union of the set of all regions provided by the DDS (only statistic nodes of the QTree) and all local data records – treated as regions with no extensions and a statistics value of 1.

Then, a peer has to determine a set $\mathbf{B_{suff}} \subseteq \mathbf{B_{all}}$ such that the worst score s is minimized and the following equation holds:

$$\sum_{B_i \in \mathbf{B_{suff}}} count(B_i) \geq N\ ,\ s := \max_{B_i \in \mathbf{B_{suff}}} s_{max}(B_i) \tag{2}$$

Based on the worst score s the peer determines all regions $\mathbf{B_{add}} \subseteq \mathbf{B_{all}} \setminus \mathbf{B_{suff}}$ that might contain data records that have a better score than s:

$$\mathbf{B_{add}} := \{B_j \in \mathbf{B_{all}} \setminus \mathbf{B_{suff}} | \, s_{min}(B_j) < s\} \tag{3}$$

Finally, the peer determines the set of relevant regions $\mathbf{B_{topN}}$ as:

$$\mathbf{B_{topN}} := \mathbf{B_{suff}} \cup \mathbf{B_{add}} \tag{4}$$

[1] The full version of this paper is available at *http://mordor.prakinf.tu-ilmenau.de/papers/dbis/2007/CoopIS07full.pdf*

The additional information that is forwarded along with the query is p_{worst}. It is the coordinates of the worst record that might be contained in the result set defined by $\mathbf{B_{suff}}$. A peer that receives p_{worst} along with the query only considers regions and local data records that are ranked better than p_{worst}.

Top-N Computation over Representatives. Let us consider each representative R to be a pair (r, B) where B is the represented region and r denotes the record that represents B. Remember that the basic algorithm for processing top-N queries needs to determine a best and a worst score for each region – $s_{\min}$ and $s_{\max}$. We can do the same for each representative by considering the boundaries of the region that it represents. Thus, we can use the same algorithm.

Minimizing the Number of Representatives. Finally, we need to minimize the number of representatives that remained in the result. For this purpose, we split up the representatives into two lists: $\mathcal{R}$ for the data records that represent the regions and $\mathcal{B}$ for the regions that are represented. Given these two lists we try to find a minimal subset of $\mathcal{R}$ that still represents all regions $\mathcal{B}$:

After having split up the representatives, $\mathcal{R}$ is sorted in descending order by the number of regions the entries could represent. We start with an empty set of chosen representatives. For each region $B \in \mathcal{B}$ we try to find an already chosen pair of (r', B') where r' can represent the merged region $B \cup B'$ without violating the approximation constraints defined by d and ε. If such a pair is found we merge the two regions and obtain a larger region that is represented by r'. If there is no such pair that has already been chosen we choose an element from $\mathcal{R}$ that could represent B. Since we have sorted $\mathcal{R}$, those $r \in \mathcal{R}$ that could represent the most regions are considered first and therefore favored. The algorithm ends after all regions of $\mathcal{B}$ are represented.

4 Introducing Constraints

Let $\mathbb{A}$ be the set of all attributes, then we define the set of user-defined constraints C as:

$$C := \{(a, n_l, n_u) | a \in \mathbb{A}, n_l, n_u \in \mathbb{R}\} \tag{5}$$

where n_l and n_u define the interval $[n_l, n_u]$ that constrains attribute a. Of course, we also support single-sided constraints by using negative and positive infinity as default for non-defined values.

In order to adapt the algorithm of Section 3 to work with constraints we have to preprocess the input data set $\mathbf{B_{all}}$ for local query processing, we obtain $\mathbf{B_{con}}$:

$$\mathbf{B_{con}} := \left\{ B_i \in B_{all} | \underset{c \in C}{\forall} \, overlaps(B_i, c) \right\} \tag{6}$$

This is the set of all regions and local data records that at least partially overlap with the data space defined by the constraints. In other words, all those records and regions are discarded that at least contradict one of the constraints.

5 Evaluation

To evaluate the algorithms presented in this paper we used the two DDS variants illustrated in Figure 1: one based on multidimensional equi-width histograms (HDDS) and one on the QTree (QDDS). We used three different setups, each is based on the same cycle-free topology of 100 peers with each peer having at most 4 neighbors. We also ran tests with other network sizes and found the same tendencies. Thus, in the following we only present our results for the network of 100 peers, where each peer provides 50 four-dimensional data records (all values restricted to $[0, 1000]$). Figure 2 shows the two-dimensional projection of the data sets. For the first setup "*Random Data, Random Distribution*" the attribute values of each data record and for all dimensions are chosen randomly from the interval $[0, 1000]$. In the second setup "*Clustered Data, Clustered Distribution*" the data of each peer is organized in a cluster. Each cluster has a diameter of 20 and is assigned randomly to a peer. For the third setup "*Anti-correlated Data, Random Distribution*" data records are chosen randomly on the line defined by the points $p_1(1000, 0, 0, 1000)$ and $p_2(0, 1000, 1000, 0)$ and offset by $[-10, 10]$.

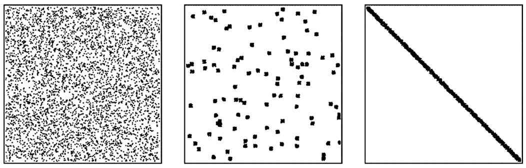
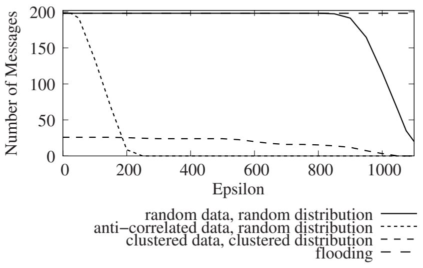
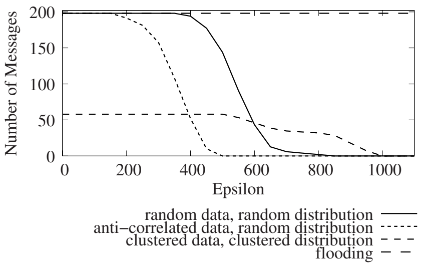


Fig. 2. Data Sets: Random (left), Clustered (middle), and Anti-Correlated (right)

In all tests the DDS are defined on all 4 attribute dimensions. QDDS use a maximum number of statistic nodes of 50 or 100 and 4 as the maximum fanout of inner nodes. HDDS use 5 or 10 buckets per dimension (i.e., 625 or 10000 for each neighbor). In all our tests we varied ε from 0 to 1100, applied the Euclidean distance as distance function, and evaluated the same test query with the same peer as initiator. In order to make use of our multidimensional index structures the top-N test query is defined on 2 attributes by the following ranking function: $attribute1 + attribute2$.



(a) Number of Messages, QDDS 50

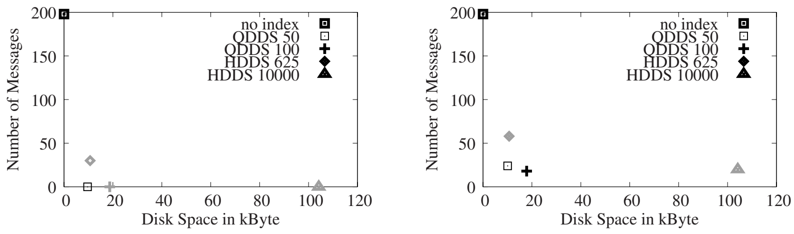


(b) Number of Messages, HDDS 625

Fig. 3. Distributed Processing of Relaxed Top-N Queries

Results. With respect to relaxation of top-N queries let us first discuss Figure 3(a) in that we used QDDS with 50 statistic nodes. As the general reduction in the number of messages for all setups indicates, in accordance to our intention the application of fuzziness reduces query execution costs: the higher ε the more cost reduction. Apart from this general tendency we can also infer the cost reduction that originates from the mere use of DDS as an ε of 0 means that relaxation is not used. In the case of clustered data in a clustered distribution the number of messages necessary to answer a query is reduced to less than 15%. The reason for this effect is that this is the best case scenario for DDS since clusters can be represented easily with low approximation error.

Figure 3(b) shows the corresponding results for HDDS. In these experiments the data of *each* neighbor was described by a histogram with 625 buckets. Remember that the QDDS were only allowed 50 statistic nodes for *all* neighbors altogether. In comparison to Figure 3(a) we see that for all setups the general tendency of message reduction with increasing ε is the same for both DDS types. In situations with little relaxation QDDS are clearly the best choice for almost all setups. But there are some situations where HDDS are the better choice: "random data, random distribution" in conjunction with higher relaxation. The reason is that in this setup there are no clusters that could easily be described by QDDS regions and this is their strength. Due to the higher number of buckets HDDS approximate the data more accurately which in turn enables a more efficient pruning and thus leads to the reduction in the number of messages. However, we still consider this a fair comparison between QDDS (50 statistic nodes) and HDDS (625 statistic nodes) because although the respective number of buckets (statistic nodes) is considerably different, their memory consumption is almost the same. As Figure 4 shows, increasing the number of statistic nodes for QDDS counteracts this problem.

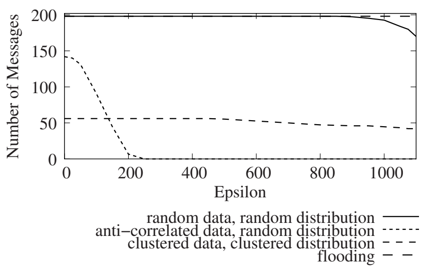


(a) Anticorrelated Data, Random Dist. (b) Clustered Data, Clustered Dist.

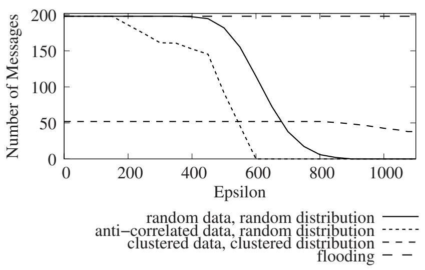
Fig. 4. Cost Benefit Analysis using a Skyline for Top-N Query Processing, $\varepsilon = 400$

Figure 4 shows the dependency of execution costs (i.e., the number of messages) on the disk space that is required to manage DDS. It shows some examples for QDDS and HDDS in two of our three setups. But what is the best choice? We have two dimensions that we want to minimize: disk space and execution costs. Thus, a skyline might help to discover the "good" choices: the black points in Figure 4 represent the skylines using an ε of 400. In both skylines QDDS dominate HDDS.

Finally, let us answer the question what happens when we additionally apply constraints. The constraints we applied restricted each queried attribute to $[500, 1000]$. One might expect that reducing the query space to a quarter might reduce execution costs as well since there is less relevant data in the network. Our results presented in Figures 5(a) and 5(b) teach us otherwise. In comparison to the full space queries, execution costs increase or stay more or less the same for 2 of 3 test scenarios and for both DDS.



(a) Number of Messages, QDDS 50



(b) Number of Messages, HDDS 625

Fig. 5. Distributed Processing of Constrained and Relaxed Top-N Queries

Only in the anti-correlated data setup with use of QDDS costs are reduced. The reason is that in the other 2 setups the result set of top-N records changes for our queries (in fact, it might be very different from the full space result) and new regions and records that have not been relevant before suddenly become interesting and have to be evaluated. This is different for the anti-correlated setup. In that case the algorithm can safely discard half of the data space without having to investigate new regions. This results in the cost reduction that we have found astonishing in the first place. As QDDS have already a rather good performance for the full space, we see this effect more clearly in Figure 5(a) than in Figure 5(b). Of course, if we reduced the data space to a very small portion of the original one, we would find a cost reduction for all setups.

6 Conclusion

In this paper, we have discussed efficient processing of rank-aware query operators in distributed environments. One of the key concepts is the use of Distributed Data Summaries (DDS) as DDS enable an efficient query routing to only those peers that are most likely to contribute to the final result. Apart from the basic strategy we have proposed the use of fuzziness such that the result does not only consist of data records but also contains representatives. Our evaluation results show that this in conjunction with DDS is an effective possibility to reduce query execution costs. Another concept that we introduced are constraints. As the evaluation shows this is only a severely limited possibility to reduce costs. The application of such constraints in general leads to a task that is not easier than the original one. However, relaxation is not restricted to distributed environments. Since the benefit was very good especially for the anti-correlated data set, future work will consider combining this technique with centralized algorithms for that especially anti-correlated data means the worst case scenario.

References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The Skyline Operator. In: ICDE 2001, pp. 421–432 (2001)
2. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: ICDCS 2002, pp. 23–32 (2002)

3. Dellis, E., Vlachou, A., Vladimirskiy, I., Seeger, B., Theodoridis, Y.: Constrained Subspace Skyline Computation. In: CIKM 2006, pp. 415–424 (2006)
4. Hose, K., Karnstedt, M., Koch, A., Sattler, K., Zinn, D.: Processing Rank-Aware Queries in P2P Systems. In: DBISP2P 2005, pp. 238–249 (2005)
5. Hose, K., Klan, D., Sattler, K.: Distributed Data Summaries for Approximate Query Processing in PDMS. In: IDEAS 2006 (2006)
6. Hose, K., Lemke, C., Sattler, K.: Processing Relaxed Skylines in PDMS Using Distributed Data Summaries. In: CIKM 2006, pp. 425–434 (2006)
7. Kung, H.T., Luccio, F., Preparata, F.P.: On Finding the Maxima of a Set of Vectors. Journal of the ACM 22(4), 469–476 (1975)
8. Lotem, A., Naor, M., Fagin, R.: Optimal Aggregation Algorithms for Middleware. In: PODS 2001 (2001)
9. Petrakis, Y., Koloniari, G., Pitoura, E.: On Using Histograms as Routing Indexes in Peer-to-Peer Systems. In: Ng, W.S., Ooi, B.-C., Ouksel, A.M., Sartori, C. (eds.) DBISP2P 2004. LNCS, vol. 3367, pp. 16–30. Springer, Heidelberg (2005)
10. Wu, P., Zhan, C., Feng, Y., Zhao, B., Agrawal, D., Abbadi, A.E.: Parallelizing Skyline Queries for Scalable Distribution. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Boehm, K., Kemper, A., Grust, T., Boehm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 112–130. Springer, Heidelberg (2006)
11. Zinn, D.: Skyline Queries in P2P Systems. Master's thesis, TU Ilmenau (2005)

Collaborative Filtering Based on Opportunistic Information Sharing in Mobile Ad-Hoc Networks

Alexandre de Spindler, Moira C. Norrie, and Michael Grossniklaus

Institute for Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
{despindler,norrie,grossniklaus}@inf.ethz.ch

Abstract. Personal mobile devices and mobile ad-hoc networks can support interesting forms of opportunistic information sharing in user communities based on spatio-temporal proximity. We show how this could be used to realise a novel decentralised collaborative filtering (CF) approach in a mobile environment.

1 Introduction

The issue of information sharing in mobile ad-hoc networks is often seen as the problem of how to ensure that users can access remote data in networks without a fixed topology and with possible disconnections. However, the ad-hoc nature of establishing network connections between personal mobile devices can be viewed as a means of sharing information *opportunistically* among members of a user community based on spatio-temporal proximity.

Although projects such as AIDE [1] and TRACE [2] have investigated the use of physical copresence as a means of forming social networks and opportunistic sharing of information, they have not considered how collaborative filtering algorithms could be adapted to base user similarity on shared social contexts. Our goal was to do exactly that and investigate the use of peer-to-peer architectures to allow users to exchange data automatically and unobtrusively based on spatio-temporal proximity.

We motivate our approach in Sect. 2 and then present our collabortaive filtering algorithm in Sect. 3. In Sect. 4, we show how our approach is equivalent to existing collaborative filtering techniques based on centralised servers. Concluding remarks are given in Sect. 5.

2 Motivation

Recommender systems based on collaborative filtering (CF) have become well-known through their use in on-line stores. The underlying assumption is that users who bought the same items in the past are likely to do so in the future. One of the first approaches developed was user-based CF [3], in which the opinions of a set of users judged to be similar to the current one are aggregated. The similarity between users is measured in terms of the extent to which their opinions about other items correlate. User-based CF has been deployed in a wide variety of application domains such as music, video and web page recommendations [4,5,6]. There are three main shortcomings of user-based CF. Firstly, the set of opinions given by a user is usually sparse and so the number of commonly rated items will be small leading to an inaccurate similarity measure. Secondly, the complexity of selecting a set of similar users grows with the number of users and items as $O(|users| \times |items|)$ leading to problems of scalability. Thirdly, when new users or items are introduced there is a lack of data on which to base recommendations.

A number of approaches address the shortcomings of user-based CF while retaining its advantages. Sarwar et al. [7] introduced the idea of item-based CF where recommendation is based on the similarity of items rather than users. User- and item-based CF are the best known representatives of so called memory-based approaches which perform filtering based on the raw data. In contrast, model-based approaches compute intermediate representations of the set of the tuples such as clusters, probability distribution functions or singular value decompositions. Model-based approaches effectively resolve the sparsity issue and render predictions more efficient and supposedly accurate.

Most collaborative filtering systems have been designed to be deployed in client-server architectures whereas only a few approaches [8,9,10] have tackled the challenges of decentralised environments. Distributed filtering research has mainly been concerned with the availability of data on client devices where network connectivity cannot be guaranteed and opinions need to be predicted. Mobile environments introduce additional challenges as limitations of size and power capacity place restricitions on computational power and human computer interface facilities. Despite the advantages of model-based CF in comparison with memory-based approaches, computing the intermediary representation emerges as a new bottleneck, in particular with regard to the limited computational power available on mobile devices. Further, although wireless connectivity is increasingly available within restricted areas such as restaurants and airports as well as public areas by means of 3G networks, area-wide connectivity is still bound to expensive communication costs, high power consumption and prone to disconnections. In contrast, devices may connect to each other in an ad-hoc peer-to-peer fashion based on short range connectivity technology such as Wi-Fi and Bluetooth. Consequently, a CF protocol for mobile environments must respect the following requirements. All computation and storage must be decentralised since a connection to a central server may not be available. Due to restricted computational and storage capacities of mobile devices, local computation must be kept simple and the required data small. Ideally, the protocol should rely on ad-hoc peer-to-peer connections only. This transient connectivity requires data exchange to be short and to consume little bandwidth. Additionally, the protocol must be delay tolerant since other peers may not always be available. Finally, since mobile devices typically feature reduced interaction facilities, user interaction should be minimal.

We believe that the notion of *shared social contexts* can be exploited to establish a similarity relationship between users. For example, if two users attend the same music concert, it is likely that they have similar musical tastes. Our initial studies carried out at an international arts festival show that this can be taken further since users who share music preferences often share preferences for other items such as festival events, films and books. As we will show in the following section, this fact can be used to reduce computing costs of CF as well as to render CF suitable for ad-hoc connectivity available in mobile environments.

3 Spatio-temporal Collaborative Filtering

The application domain of CF contains users consuming items and expressing opinions about these items. Based on these, a collaborative filtering system predicts their opinion about items unknown to them. Opinions are tuples of the form $(user, item, value)$ which can be seen as a directed weighted edge in a graph, pointing from a user node to an item node and weighted with a rating value. Thus, a set of tuples defines a directed graph $G = (U \cup I, E)$ where U is the set of nodes representing users, I the set of item nodes and E the set of directed weighted edges pointing from nodes in U to nodes in I.

User-based CF processes a fundamental query by first computing similarities among users and selecting those judged to be similar. Then the ratings of target items by these users are aggregated. In order to include user similarities, we augment the previously defined graph with undirected edges connecting two users and weighted with their similarities. Thus, the set of edges E is now composed of $E_r \cup E_s$ where E_r contains the rating edges and E_s the similarity edges. As proposed by Mirza et al. [11], $G_s = (U, E_s)$ represents a *social network graph* while $G_r = (U \cup I, E_r)$ refers to the *rating graph*.

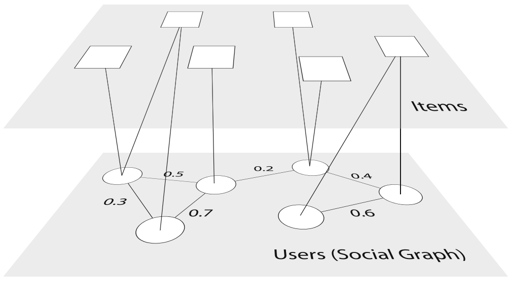


Fig. 1. Social- and rating graph

Figure 1 shows an example graph composed of a social and rating graph. The vertices on the bottom layer represent users and the ones on the top layer items. Edges connecting users are weighted with the similarity of the adjacent users. For clarity, we omit the weights of the edges connecting a user to an item representing the rating value.

In our approach, the selection of users is performed implicitly and without any prior similarity computations. We introduce the concept of *spatio-temporal* *proximity* which forms the basis for our selection of similar users. In the case of social contexts formed around consumable items, user consumption of an item means that their location matches the location of the item for a specific period of time. Some items such as restaurants or bars can be consumed at any time within predefined opening hours and the duration of consumption can be anything from the time to drink a glass of wine up to eating a dinner. In contrast, items such as comedy shows or theatre plays can be consumed only during a specific time period and the duration is usually well defined. We will refer to these two kinds of items as *location* and *event* items, respectively. Note that event items may happen only once or be repeated periodically. All items have in common the fact that if users meet while consuming them, they stay in each other's vicinity for longer than if they would pass each other in the street by chance.

The history of item consumption of a particular user u_a can be regarded as a set of *item consumption tuples* of the form $(loc_i, [t_k, t_l])$ where each tuple contains two entries. The first entry identifies a location loc_i particular to an item. This location represents an area in which the item can be consumed. The second one delimits a period of time $[t_k, t_l]$ during which the item was consumed. Consequently, the history $H(u_a)$ of a user u_a can be written as $H(u_a) = \{(loc_1, [t_1, t_2]), (loc_2, [t_3, t_4]), \ldots\}$. The condition for item consumption tuples to be equal is $(loc_i, [t_k, t_l]) = (loc_j, [t_m, t_n]) \iff (loc_i = loc_j) \wedge ([t_k, t_l] \cap_t [t_m, t_n] \geq p)$ where we define $\cap_t$ as a temporal intersection of two time periods. The condition $[t_k, t_l] \cap_t [t_m, t_n] \geq p$ holds if the time periods overlap for a duration of at least p. The first component $(loc_i = loc_j)$ accounts for spatial proximity while the temporal intersection accounts for temporal proximity.

The user similarity $P_{loc,t}$ resulting from spatio-temporal proximity between two users u_a and u_b can be expressed as

$$P^{\mathbb{N}}_{loc,t}(u_a, u_b) = \begin{cases} 1 & \text{if } H(u_a) \cap H(u_b) \neq \emptyset \\ 0 & \text{else.} \end{cases}$$

This is a binary similarity measure in the sense that users are evaluated to be similar only if they have at least one tuple of their consumption history in common. We use $P^{\mathbb{N}}_{loc,t}(u_a, u_b)$ as a condition for the users $u_a \in U$ and $u_b \in U$ to be connected by a similarity edge $(u_a, u_b) \in E_s$ in the social graph G_s. The resulting social graph corresponds to a copresence community used by Lawrence et al. [1] to disseminate information since spatio-temporal proximity is a necessary and sufficient condition for users to have their devices connected.

We can refine this similarity taking into consideration the level of spatio-temporal proximity among users. Based on the fact that users consuming the same items are similar, it is obvious that the more often users consume the same item, the more similar they are. This calls for a continuous similarity measure $P^{\mathbb{R}}_{loc,t}$ that takes into account the number of common simultaneous item consumptions as opposed to the binary measure proposed before.

$$P^{\mathbb{R}}_{loc,t}(u_a, u_b) = \begin{cases} \frac{|H(u_a) \cap H(u_b)|}{max(|H(u_a)|, |H(u_b)|)} & \text{if } H(u_a) \neq \emptyset \\ 0 & \text{else} \end{cases}$$

This measure allows us to assign a weight to a similarity edge created based on the binary measure. Note that if it evaluates to zero, the respective users are not connected in the graph while it never evaluates to zero if they are connected.

We now describe our CF approach in terms of a formal description of the algorithm running on a single mobile device as shown in Figure 2. For this discussion, we assume the existence of three library functions. WAIT(p) causes the algorithm to pause for a time period of p, TRANSMIT($Peer, M$) transmits a set of edges M to a remote peer $Peer$. This transmission will be translated to a call of the function RECEIVE(M) on the remote peer where M corresponds to the second argument of the transmission function. INCREASE-WEIGHT($Peer$) retrieves the edge $(u_{local}, u_{remote}) \in E_s$ where u_{remote} denotes the user node representing the argument $Peer$ and increases its weight in order to update the respective continuous proximity value.

```
MAIN-LOOP()
1  $N \leftarrow \emptyset$
2  while $run = \top$
3  do $N_{current} \leftarrow$ SCAN()
4      $N_{new} \leftarrow N_{current} - N$
5      for $\forall$ $Peer \in N_{new}$
6      do SEND($Peer$)
7          INCREASE-WEIGHT($Peer$)
8      $N \leftarrow N_{current}$
```

```
SEND($Peer$)
1  $M \leftarrow \emptyset$
2  for $\forall$ $(u_{local}, i) \in E_r$
3  do $M \leftarrow M \cup \{(u_{local}, i)\}$
4  WAIT($p$)
5  TRANSMIT($Peer, M$)
```

```
RECEIVE($M$)
1  for $\forall$ $(u_{remote}, i) \in M$
2  do $E_r \leftarrow E_r \cup \{(u_{remote}, i)\}$
```

Fig. 2. Collaborative filtering algorithm

While a peer is active, i.e. $run = \top$, the main loop simply scans the environment periodically and maintains a set N of peers in the vicinity. For every remote peer $Peer$ in the vicinity, the method SEND($Peer$) is called to send all ratings made by the local user to the remote peer. This method runs as a thread per remote peer in order to be non-blocking. Note that these ratings will only be sent after a delay of length p, the parameter introduced above to determine the equality of two rating consumption tuples. If the remote peer has left the vicinity of the local peer during this time period, the tuples will not be sent by the TRANSMIT($Peer, M$) function to avoid exchanges during a transient encounter. Once the rating tuples have been sent to all new peers in the vicinity, the set of peers in the vicinity is updated to remove peers that have left. Whenever a local peer receives a set of tuples from a remote peer, RECEIVE(M) is called and these tuples are added to the set of tuples stored locally.

Finally, rating values from similar users about the target item are aggregated. The most common approach is to compute the average. To do so, we select all incoming edges of the node representing the target item and compute the average of their weights. We also take into account the degree of similarity as expressed by the continuous proximity measure. $P^{\mathbb{R}}_{loc,t}(u_a, u_b)$ establishes a ranking of the users according to their similarity to the user denoted by the first argument. A user u_a is more similar to a user u_b than to another user u_c if $P^{\mathbb{R}}_{loc,t}(u_a, u_b) > P^{\mathbb{R}}_{loc,t}(u_a, u_c)$. Consequently, if we are to predict a rating value for a requesting user u_r about a target item it_t, we compute the average of the rating values contained in G_r, each weighted with the respective edge weights in G_s. When computing this weighted average, we only need the continuous proximity values for the rating user to all other users in the local graph. The similarity between other users does not affect the aggregation and thus no continuous proximity information needs to be passed on when ratings are exchanged.

4 Equivalence to Existing Algorithms

As explained in the previous section, users of our recommender system exchange tuples when they are in spatio-temporal proximity. Each user maintains a graph G_{local} where the nodes in U represent users previously met and the nodes in I represent all items rated by these users or the local user. In this section we first explain why such a local graph is sufficient to perform user-based collaborative filtering. Secondly, we show that the resulting algorithm resolves scalability issues for which user-based approaches have frequently been criticised.

We first look at a simple form of traditional user-based CF where rating values are set to 1 if a user has consumed an item and 0 otherwise. For example, the Amazon online store interprets the purchase of an item as an expression of a binary opinion about it. Thus, each user is represented by a binary vector containing entries for all items. A server maintains the set of user vectors based on which ratings are predicted. The similarity between two users is computed as the number of vector entries both have set to 1. The prediction is the result of aggregating the ratings of all users about the target item, each weighted with the similarity between the requesting and rating user. Consequently, the prediction is based on the set of users that have consumed at least one item which the requesting user has also consumed. All other users are not included because their ratings are weighted with a zero-valued similarity.

A user vector is a set of rating tuples where the user entry contains the represented user. The tuples stored on the server define a graph G_{global} which, in contrast to a local graph, includes all participating users and items consumed by any user. Therefore, a local graph is a subgraph of the global graph while the global graph is a union of all local graphs. In fact, a local graph belonging to a particular user u_i can be extracted from the global graph as follows. We use superscript notations g and l to indicate that a node or edge set belongs to the global or local graph, respectively. An edge is denoted as (p, q) where p and q are the adjacent nodes. Finally, $w_{(p,q)}$ refers to the weight of an edge (p, q).

$$U^l = \{u \mid u \in U^g \wedge (u_i, u) \in E^g_s \wedge w_{(u_i,u)} > 0\} \cup u_i \tag{1}$$

$$I^l = \{i \mid i \in I^g \wedge (u, i) \in E^g_r \wedge u \in U^l\} \tag{2}$$

$$E^l_r = \{(u, i) \mid (u, i) \in E^g_r \wedge u \in U^l \wedge i \in I^l\} \tag{3}$$

$$E^l_s = \{(u_i, u) \mid (u_i, u) \in E^g_s\} \tag{4}$$

Equation 1 states that we take all users in U^g which are connected to u_i by a similarity edge with a weight greater than zero. Equation 2 selects all items that are connected to a user selected in Eq. 1. Equation 3 selects all rating edges whose adjacent user and item have been selected by the previous two equations. Finally, Eq. 4 accounts for the fact that similarity edges are not exchanged. Thus, only similarity edges between u_i and the other users are selected. In Fig. 3, we highlight the local graph as part of the global graph. Nodes and edges not belonging to the local graph are drawn with a dashed line.

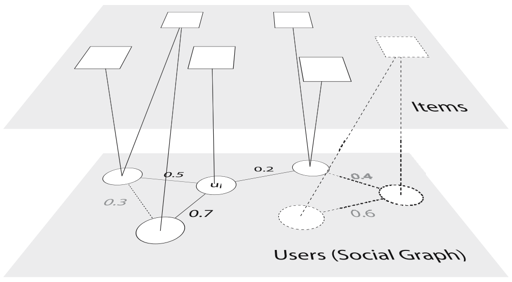


Fig. 3. Local graph as a subgraph of the global graph

Simple traditional CF outlined above predicts a rating for a requesting user u_r about a target item i_t as

$$\frac{1}{|(u,i_t) \in E_r^g : (u_r,u) \in E_s^g|} \sum_{(u,i_t) \in E_r^g} w_{(u,i_t)} \cdot w_{(u_r,u)} \tag{5}$$

where the aggregation is a weighted average of the ratings. Now we want to show that all rating and social edges included in the aggregation also exist in the local graph belonging to u_r. The underlying intuition is that users from whom ratings are aggregated have in common the fact that they consumed items also consumed by the requesting user. Hence, if users exchange their own ratings whenever they consume the same item, the set of users from whom opinions are collected and thus are available in the local graph is equivalent to the set of users selected in the global graph by traditional user-based CF. In order to prove this equivalence, we have to show that all rating and social edges included in the sum in Eq. 5 also exist in the local graph. This is obvious for the social edges because all edges $w_{(u_r,u)} \in E_s^g$ have been selected by Eq. 4 and, since we are considering the local graph belonging to u_r, it holds that $u_i = u_r$. In order to simplify this proof of equivalence, we can now leave out the weighting of each rating. Therefore we rewrite Eq. 5 as

$$\frac{1}{|(u,i_t) \in E_r^g : (u_r,u) \in E_s^g|} \sum_{(u,i_t) \in E_r : (u_r,u) \in E_s^g \wedge w_{(u_r,u)} > 0} w_{(u,i_t)} \tag{6}$$

where the condition of the sum ensures rating edges are included only from rating users that have a non-zero similarity to the requesting user. Now it is apparent that the rating edges included in the aggregation are also contained in the local graph since E_r^l is extracted from E_r^g by applying Eq. 1, 2 and 3 consecutively, while the selection criteria of the sum is equivalent to Eq. 1.

Since a local graph contains all edges used by traditional user-based CF for rating predictions based on the global graph, the same results can be obtained from the local graph alone thereby eliminating the need for a central server to store all user vectors to compute similarities between users which is considered the main bottleneck in traditional CF.

5 Conclusions

We have presented a technique for user-based collaborative filtering that exploits an opportunistic mode of information sharing resulting from ad-hoc peer-to-peer networking. Only users in spatio-temporal proximity are able to exchange ratings and we have shown how this provides a natural filtering based on social contexts. The resulting selection of similar users renders the computation of similarities and selection of most similar users unnecessary which resolves sparsity and scalability issues frequently associated with user-based collaborative filtering.

References

1. Lawrence, J., Payne, T.R., Roure, D.D.: Co-presence Communities: Using Pervasive Computing to Support Weak Social Networks. In: Proc. Intl. Workshop on Distributed and Mobile Collaboration (2006)
2. Counts, S., Geraci, J.: Incorporating Physical Co-presence at Events into Digital Social Networking. In: Extended Abstracts on Human Factors in Computing Systems (2005)
3. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In: Proc. Conf. on Computer Supported Cooperative Work (1994)
4. Shardanand, U., Maes, P.: Social Information Filtering: Algorithms for Automating "Word of Mouth". In: Proc. Intl. Conf. on Human Factors in Computing Systems (1995)
5. Hill, W., Stead, L., Rosenstein, M., Furnas, G.: Recommending and Evaluating Choices in a Virtual Community of Use. In: Proc. Intl. Conf. on Human Factors in Computing Systems (1995)
6. Terveen, L.G., Hill, W.C., Amento, B., McDonald, D., Creter, J.: Building Task-Specific Interfaces to High Volume Conversational Data. In: Proc. Conf. on Human Factors in Computing Systems (1997)
7. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Item-based Collaborative Filtering Recommendation Algorithms. In: Proc. Intl. Conf. on World Wide Web (2001)
8. Wang, J., Pouwelse, J., Lagendijk, R.L., Reinders, M.J.T.: Distributed Collaborative Filtering for Peer-to-Peer File Sharing Systems. In: Proc. Symp. on Applied Computing

9. Miller, B.N., Konstan, J.A., Riedl, J.: PocketLens: Toward a Personal Recommender System. ACM Trans. Inf. Syst. 22(3), 437–476 (2004)
10. Tveit, A.: Peer-to-peer Based Recommendations for Mobile Commerce. In: Proc. Intl. Workshop on Mobile Commerce
11. Mirza, B.J., Keller, B.J., Ramakrishnan, N.: Studying Recommendation Algorithms by Graph Analysis. J. Intell. Inf. Syst. 20(2), 131–160 (2003)

Policy-Based Service Registration and Discovery*

Tan Phan[1], Jun Han[1], Jean-Guy Schneider[1], Tim Ebringer[2], and Tony Rogers[2]

[1] Faculty of ICT, Swinburne University of Technology, 3122 Hawthorn, Australia
{tphan, jhan, jschneider}@ict.swin.edu.au
[2] CA Labs, CA (Pacific), Building 10, Level 2, 658 Church Street, 3121 Richmond, Australia
{Tim.Ebringer, Tony.Rogers}@ca.com

Abstract. The WS-Policy framework has been introduced to allow policy to be expressed and associated with Web Services thereby enabling organizations to manage the quality of their services. How the specified polices are kept consistent with the organization's regulations, and how to match service and client policies requirements for effective service discovery, are issues yet to be addressed. In this paper, we present a new approach that allows for the automatic verification and matching of policies, using a service registry that serves as a policy storage and management facility, a policy checkpoint during service publication and as a policy matchmaker during service discovery. We extend WS-Policy with a policy conformance operator for policy verification and use WS-Policy Intersection for policy matching. We develop a policy information model and policy processing logics for the registry. An implementation of a policy-enabled service registry is also introduced.

1 Introduction

Requirements for quality and standard compliance are often specified in the form of policies about the non-functional requirements of various components in an organization's IT infrastructure. In the Web Services context, the WS-Policy [1] framework provides a way to describe the policies regarding Web Services in a machine readable form; this allows for automatic policy enforcement via policy-aware clients. At present, policies are associated with Web Services in various ways and there is no automatic mechanism to guarantee that the policies specified are consistent with the organization or application's specific requirements. There is, therefore, a need for reliable techniques to evaluate policies on a large number of services and a final check point for policy conformance before the services are published and made available to the clients.

Another concern is service discovery which, at present, focuses more on functional and less on non-functional aspects of the services. WS-Policy allows a service to advertise its capabilities and specify its requirements. Unfortunately, there is no easy way for the client to have access to the policy information; this typically requires some out of band communication. Other approaches such as WS-MetadataExchange

* This work is supported by the Australian Research Council and CA Labs.

[4] have suggested ways of adding service meta-data such as policy information into service endpoint description but this still requires the client to know the service endpoint's address.

In this paper, we present an approach to address these issues using a service registry that holds policy information. We argue that a verification unit should reside inside the service registry to verify the services for policy conformance when they are published. We also propose that service clients should be able to indicate their policy requirements as part of the service query. We thus present a management model and techniques to enable automatic policy verification and matching. A prototype tool has also been implemented to demonstrate the approach.

2 Background

The WS-Policy framework comprises a set of specifications that together offer "mechanisms to represent the capabilities and requirements of Web Services as policies" [15]. The framework includes the WS-Policy language [4], which provides a simple and extensible notation to combine the various kinds of policy assertions and form policy descriptions, and the WS-PolicyAttachment specification [3] which specifies how to associate a policy with Web Services entities (services, endpoints, operations, and messages). Various domain-specific standards have also been defined to allow for the expression of policy assertions in individual domains like WS-SecurityPolicy [12], WS-ReliableMessagingPolicy [5], and MTOM [10]. Policy-aware tools such as WSE [13] can generate code to perform policy enforcements automatically. WS-Policy is by itself a simple declarative language with the following normalized structure.

```
<wsp:Policy> <wsp:ExactlyOne>
  (<wsp:All>   (<Assertion>   …   </Assertion>)*   /wsp:All>)*
</wsp:ExactlyOne> </wsp:Policy>
```

Effectively, in its normal form a WS-Policy policy is a logical XOR of the contained policy alternatives with each policy alternative being a logical AND of the assertions contained as seen in the following expression:

$P = XOR(AND(A_{11},...,A_{1m_1}),...,AND(A_{n1},...,A_{nm_n}))$: Where P is a policy expression; A_{ij} ($0 \le i \le n, 0 \le j \le m$) is the jth policy assertion in the ith policy alternative of the expression. Any WS-Policy policy expressions can be normalized into the above form. Therefore, for the sake of simplicity, and without losing generality, in this paper we treat all WS-Policy policy expressions as if they are in their normal form.

A service registry holds service metadata for the registration and discovery of Web services. Registries are characterized by rich metadata management and rich query capabilities. There are two popular registry specifications: UDDI [7] and EbXML Registry [9]. The two specifications were originally created for standardizing inter-organizational service registry products. They are now adopted more for intra-organizational registries due to the trust and privacy issues related to service registration and discovery spanning multiple organizations. At present, neither UDDI nor EbXML Registry has direct support for policy processing.

3 Enabling Policy-Based Service Registration and Discovery

We advocate the use of a service registry to support policy verification and policy-based discovery. A service registry is where all service metadata is registered and stored making it suitable for capturing and processing policy information. Existing service metadata management mechanisms in the registry can be leveraged to support the creation, publication, modification and removal of policy.

In our approach, a typical service registry's data model is enhanced with a policy information model to represent policy information. Two additional units, the `PolicyValidator` for service policy verification at publishing time and the `PolicyEnabledQueryManager` for policy matching at the service discovery time, are added to the registry as can be seen in Figure 1.

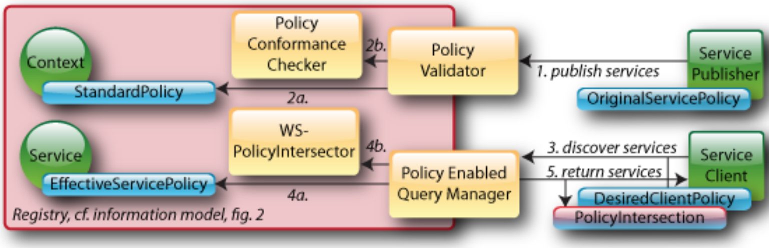


Fig. 1. A registry-based model for policy registration and discovery

Contexts are defined within a registry to represent organizations, applications or development projects. Policy can be specified per context to represent all the common requirements, such as those about security like authentication, authorization, message encryption and signing, that any entity under that context must follow. When a service is published into a context with a policy, that policy must be verified against the policy requirements of the context. The service will only be stored in the registry when the policy conforms to the requirements. When the service policy or context policy is updated, policy will be revalidated. This guarantees that only the services with appropriate policies are made available for consumption by clients.

When a service client looks for a service in a registry, the client may only be interested in a set of services that, apart from satisfying the functional requirements, also support certain policy requirements. Client side policy requirements are conveyed by sending the registry a description of the desirable policy as part of the selection criteria of the service discovery. Only services that support the desired client policies are returned. With the use of a policy-aware registry non-functional (policy-based) and functional discovery can be achieved at the same time.

3.1 Policy Information Model

To support the storage and matching of service policy information inside the registry, we use the abstract information model presented in Figure 2.

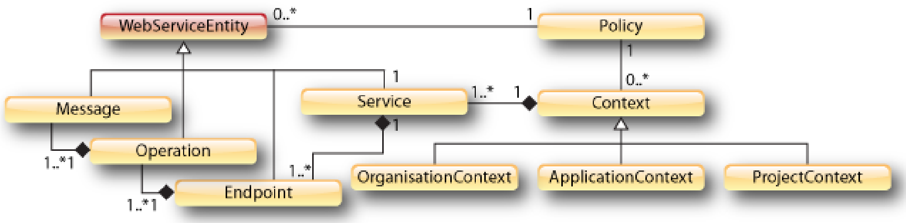


Fig. 2. UML Class diagram for policy information model

This model depicts the relationship between Web Services entities, the context that the entities are deployed to, and the policies that are specified for the entities and the context. Web Services entities are modeled following WSPolicy-Attachment [3]. Essentially, a service consists of one or many physical endpoints, with each endpoint consisting of a collection of operations, and each operation in turns containing a collection of messages. They are all referred to as `WebServicesEntity`.

`Context` models an aggregation of `RegistryEntities` with some common settings and typically represents an organization, an application, or a development project. A Service must exist under one context, which is also the context of the service's endpoints, operations, and messages. A context is associated with one policy description (called the `StandardPolicy`) which encapsulates all the policy requirements that objects to be deployed into the context must conform to. Policy represents a policy description, which might be the (WS-Policy) *merge* [4] of multiple (WS-Policy) policy documents. The association between a Web Services entity and a policy is the association between the entity itself and its *effective* policy. The *effective* policy of a given Web Services entity is a *merge* of the policy that is attached to the entity itself and any policies that the entity inherited from its container entities, following the mechanism specified in WS-PolicyAttachment [3].

3.2 Enabling Policy-Based Service Registration

A service published into the registry must have an associated context which is indicated in the service publishing message. Context policy represents policy requirements that apply to *every* entity deployed under the context. The service provider might have different policy requirements for each of the endpoints, operations or messages in the service he publishes so different policies might be specified for the services, endpoints, operations, and messages themselves. In this case, the *effective* policy of each of these entities must conform to the *`StandardPolicy`* for the service to be considered conforming to the requirements of the context.

Policy conformance: Policy conformance verification has a non-commutative nature. That is, the fact that a policy P_1 conforms to another policy P_2 does not imply that P_2 conforms to P_1. Currently, WS-Policy Intersection [4] is referred to as the WS-Policy's operator for policy conformance checking [15]. Being a commutative operator, WS-Policy-Intersection is unsuitable for the checking. Below, we propose a WS-Policy Conformance operator which is non-commutative and asymmetric. We first start with a general definition for policy conformance

Definition 1: *A policy P_1 is said to conform to a policy P_2 when the fact that one entity satisfies P_1 implies that the entity also satisfies P_2.* This implies that P_1 specifies equal or more stringent requirements and/or equal or more capabilities than P_2 does.

Policy conformance for WS-Policy: Because, in normal form, a WS-Policy policy is an XOR of different policy alternatives, Definition 1 can then be refined as follows.

Definition 2: *In their normal form, a WS-Policy expression P_1 is said to conform to another WS-Policy expression P_2 when the fact that an entity satisfies/supports an alternative in P_1 implies that the entity satisfies/supports one alternative in P_2.*

Definition 3: *A WS-Policy policy alternative PA_1 is said to conform to another policy alternative PA_2 when the fact that one entity satisfies PA_1 implies that the entity also satisfies PA_2.* A WS-Policy alternative is a logical AND of policy assertions, therefore the support for *any assertion* in PA_2 above is implied by the support of *all the assertions* in PA_1. At the assertion level, we rely on a domain-specific conformance function to determine whether a set of policy assertions collectively satisfy another assertion.

The following is an algorithm for verifying policy conformance in WS-Policy derived from the definitions given above. The algorithm assumes the presence of a domain-specific conformance function for any assertions within each of the policy. In the absence of the domain-specific conformance function, the default rule, which is *an assertion conforms to and only to itself*, is applied.

Policy conformance algorithm: Given two normalized policies P_1, the policy that needs to be verified, and P_2, the standard policy:

1. If P_1 is empty (P_1 has no alternative, meaning the set of behaviors accepted by P_1 is empty) then P_1 conforms to P_2 regardless of P_2's structure
2. If P_2 is empty then P_1 does not conform to P_2 unless P_1 is also empty
3. P_1 conforms to P_2 when *each* alternative in P_1 conforms to one alternative in P_2.
4. Given a policy alternative A_1 of P_1 and a policy alternative A_2 of P_2
   a. If A_2 is empty (A_2 has no assertion meaning it can accept any behavior) then A_1 always conforms to A_2
   b. If A_1 is empty then A_1 does not conform to A_2 unless A_2 is also empty
   c. A_1 conforms to A_2 when *each* policy assertion in A_2 is satisfied by *all assertions* in A_1 collectively, using a predefined domain-specific conformance function

Domain-specific conformance function: We assume that policy assertion authors (such as the WS-SecurityPolicy committee members) supply, together with the assertions, the domain-specific policy conformance function. The conformance function for a domain should be able to determine, within the domain, whether a logical AND of a set of assertions conforms to an arbitrary assertion; this is required because sometimes an individual assertion might not conform to another one while a logical AND of assertions might. For example, if we have a policy assertion S_1 requiring the encryption of the entire SOAP message (i.e. encrypting the *envelop* element), an assertion S_2 requiring *header* encryption, and an assertion S_3 requiring *body* encryption, then neither S_2 nor S_3 alone conforms to S_1, but S_2 and S_3 collectively would conform to S_1.

Policy verification processing: The `PolicyValidator` unit of the registry extracts the service's context from the service publication message and, based on the context, retrieves the context's *`StandardPolicy`*. It also extracts the service's attached *`OriginalServicePolicy`* and forwards the two policies to the `PolicyConformanceChecker` for verification using the presented algorithm.

In case separate policies are specified for endpoints, operations, and messages, the registry will first calculate the *effective* policies for each of the entities. The calculated *effective* policies of the endpoints, operations and messages of the published service are then verified against the *`StandardPolicy`*. Only when they all conform to *`StandardPolicy`* is the service allowed to be published under the context.

3.3 Enabling Policy-Based Service Discovery

Client side policy requirements are indicated as part of the service discovery query, which can span multiple contexts. Also, the client might have specific policy requirements for service, endpoint, operation, or message levels. In this case, it can supply separate policies and uses the mechanisms specified in WSPolicyAttachment [3] to apply the policies to the endpoint, operation or message scope as needed.

Policy matching is based on the compatibility of the client policy and the service policy. To determine policy compatibility we use the Policy Intersection algorithm defined in WS-Policy. WS-Policy's Policy Intersection is designed to check whether one side of the interaction will support the conditions indicated by the other side and what is the policy that will be mutually manifested on the wire during the interaction. This logic is thus suitable for policy discovery as the client can use WS-Policy Intersection to determine whether a target service supports its own policy requirements. We introduce in brief the policy compatibility logics defined in WS-Policy Intersection.

Policy compatibility: According to Policy Intersection, two policies P_1 and P_2 are said to be compatible when they have a non-empty intersection, meaning that there exists at least an alternative in P_1 that is compatible with an alternative in P_2 and vice versa. A policy alternative A_1 is compatible with a policy alternative A_2 when, for each assertion in A_1, there exists a compatible assertion in A_2 and vice versa. Two policy assertions are said to be compatible with each other if the presence of one implies (in a domain-specific way) the support for the other. In the absence of a domain-specific assertion compatibility function, a policy assertion S_1 is said to be compatible with a policy assertion S_2 when they specifies the same type of capability or requirement (having the same Qualified Name) and if one assertion has a nested policy then the other must also have the same nested policy.

Policy matching processing: For each functionally matched service found, the PolicyEnabledQueryManager resolves the service's associated policy description (the service's *EffectivePolicy*) and forwards it to the PolicyIntersector to perform the intersection. If the intersection policy is non-empty, meaning the service and client policies are compatible, the service is then added to the returned set. In case separate policy requirements are specified for the service, endpoint, operation, and message levels, the registry, upon receiving the client policies, will perform the calculation for the *effective* policies for each level of the client requirements. For each target service, it also calculates the *effective* policies for the service, its endpoint, operations and messages and then performs the policy matching at these levels accordingly. Only when matching is achieved at *all* the specified levels, is the service considered having a policy that matches client's requirements.

3.4 Example

A bank creates the following standard policy for the CorporateContext in its internal registry, which requires SOAP *body* encryption for every Web Services message

```
<wsp:Policy…> <wsp:ExactlyOne> <wsp:All> <sp:EncryptedElements>
          <sp:XPath>/S:Envelope/S:Body</sp:XPath>
</sp:EncryptedElements> </wsp:All> </wsp:ExactlyOne> </wsp:Policy>
```

A developer X developed a service named InterestRateQuote. He believes the service is critical and thus requires the encryption of the entire SOAP message *envelope*:

```
<wsp:Policy…> <wsp:ExactlyOne> <wsp:All> <sp:EncryptedElements>
          <sp:XPath>/S:Envelope</sp:XPath>
</sp:EncryptedElements> </wsp:All> </wsp:ExactlyOne> </wsp:Policy>
```

For another service ExchangeRateQuote, X believes the service can function properly with no further requirements so a policy with only one empty alternative (meaning any behavior is accepted) is supplied:

```
<wsp:Policy…>
        <wsp:ExactlyOne> <wsp:All/> </wsp:ExactlyOne>
 </wsp:Policy>
```

When the two services InterestRateQuote and ExchangeRateQuote are published into the CorporateContext of the registry, only InterestRateQuote is accepted because the services' policy, which requires SOAP *envelop* encryption, is stronger than and thus conforms to the standard requirement of SOAP *body* encryption while ExchangeRateQuote's policy requirements is weaker than that of the context.

Another developer Y wants to query the registry to find a live base-interest-rate quote service that supports SOAP *header* encryption as indicated in his desirable policy:

```
<wsp:Policy…> <wsp:ExactlyOne> <wsp:All> <sp:EncryptedElements>
          <sp:XPath>/S:Envelope/S:Header</sp:XPath>
</sp:EncryptedElements> </wsp:All> </wsp:ExactlyOne> </wsp:Policy>
```

When Y submits a query to the registry with this desired policy, the registry returns `InterestRateQuote` because the service's and Y's policy are compatible (SOAP *envelop* encryption implies the support for SOAP *header* encryption and vice versa).

4 Implementation

We have implemented (in Java) the support for policy on top of the CA eTrust UDDI 3.0. We implemented an initial mapping of the policy information model to UDDI version 3 data structures as shown in the following figure.

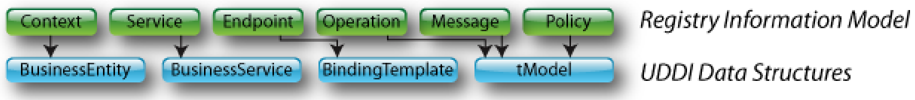


Fig. 3. Mapping Registry Information Model to UDDI data structures

A high-level architecture of the software is depicted in Figure 4 below. In this figure, the components colored in grey are the added/modified components and the rest are those existing in the current CA eTrust UDDI. Specifically we have modified the `saveServiceImpl` and `findServiceImpl` components of the registry, which are responsible for registering and discovering services respectively, to accommodate the support for policy processing following the logics described in Section 3. We added the implementation for the policy conformance and WS-Policy intersection algorithm (Section 4) to the open source Apache WS-Policy library – Apache Neethi [1] and use the library as the `PolicyIntersector` and `ConformanceChecker`.

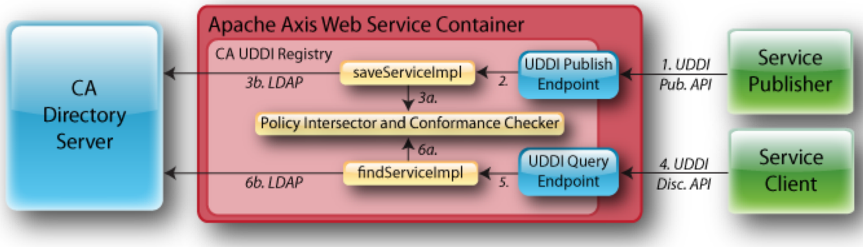


Fig. 4. eTrust UDDI's support for policy

To enable the attachment of policy descriptions to UDDI service publishing and querying messages we define a UDDI *tModel* key for policy information. When the service publisher publishes a service or when the service client queries for a service, they supply the URI to the associated policy description as the value in a key-value pair entry – the UDDI *keyedReference*, with the key of the entry being the predefined policy *tModel,* in the UDDI *categoryBag* of the service.

5 Related Work

There has been a body of work in the area of enhancing service discovery with non-functional matching. UDDIe [14] attempts to extend UDDI with the notion of 'blue pages' for enabling service discovery based on user-defined properties like Quality of Service (QoS) that a service can provide, or the methods available within a service. Similarly, WSLA framework in [11] provides the support for a Service-Level Agreement between a service provider and a potential requestor which allows clients to be able to find services that satisfy their QoS requirements. The main difference between our work and these works is none of the works above discusses how to ensure the properties specified for a service are valid and conforms to the organization's requirements.

The Ponder framework, which includes a specification language, a management model [8] allows for the policy-based management of distributed systems. However, the work focuses more on transport-level access control and QoS management for telecommunication networks rather than for application-level management of SOA systems. Another policy language based on decision tree has been defined in [16] but only limited to specifying QoS parameters for SOA components. This work also advocates the use of a service registry as the storage mechanism for policy information, but the work does not provide any information on how the policy is to be stored in registry and how the parties involved in policy verification and management can interact with the registry.

6 Conclusion and Future Work

In this paper, we have presented a registry-based framework and the associated techniques to enable the automatic verification of service policy information when a service is registered and the automatic matching of service and client policy information when the service clients query for a service respectively. Verification and match making can be done at the endpoint, operation or message levels of the service. Policy verification is based on a conformance operator we developed for WS-Policy, while policy matching is based on WS-Policy Intersection. In future work, we plan to define a framework for policy-based runtime service and application management using registry.

Reference

1. The Apache Software Foundation (2007). Apache Neethi 2.0. (June 2007), http://ws.apache.org/commons/neethi/index.html
2. Bajaj, S., et al.: Web Services Policy Framework 1.2. W3C (April 2006)
3. Bajaj, S., et al.: Web Services Policy Attachment 1.2. W3C (April 2006)
4. Ballinger, K., et al.: Web Services Metadata Exchange 1.1. IBM, BEA Systems, Microsoft, SAP, AG, CA, Sun Microsystems, and webMethods (August 2006)
5. Bilorusets, R., et al.: Web Services Reliable Messaging Protocol 1.0. IBM, BEA Systems, Microsoft, and TIBCO Software (Febuary 2005)
6. Box, D., et al.: Web services addressing (WS-Addressing). W3C (August 2004)
7. Clement, L., et al.: Universal Description, Discovery, and Integration 3.0. OASIS (October 2004)
8. Damianou, N.: A Policy Framework for Management of Distributed Systems. PhD Thesis, Imperial College, London (2006)
9. Fuger, S., et al.: EbXML Registry Information Model 3.0 and EbXML Registry Service and Protocol 3.0. OASIS (May 2005)
10. Gudgin, M., et al.: SOAP Message Optimization Transmission Mechanism 1.0. W3C (January 2005)
11. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreement for Web Services. J. of Network and Systems Management (2003)
12. Lawrence, K., et al.: Web Services Security Policy 1.2. OASIS (2005)
13. Microsoft (2005). Web Services Enhancement 3.0. Released in (July 2005), http://msdn2.microsoft.com/en-us/webservices/aa740663.aspx
14. ShaikhAli, A., Rana, O.F., Ali-Ali, R., Walker, D.V.: UDDIe: an extended registry for Web Services. In: Proc. Application and Internet Workshop 2003, Orlando, FL, USA (2003)
15. Vedamuthu, A., et al.: Web Services Policy 1.5 - Primer. W3C (June 2007)
16. Wang, C., Wang, G., Chen, A., Wang, H., Pierce, Y., Fung, C., Uczekaj, F.: A Policy-Based Approach for QoS Specification and Enforcement in Distributed Service-Oriented Architecture. In: SCC 2005. Proc. 2005 IEEE Int'l Conf. on Services Computing, FL, USA, IEEE Computer Society Press, Los Alamitos (2005)

Business Process Quality Metrics: Log-Based Complexity of Workflow Patterns*

Jorge Cardoso

Department of Mathematics and Engineering,
University of Madeira, Funchal, Portugal
jcardoso@uma.pt

Abstract. We believe that analysis tools for BPM should provide other analytical capabilities besides verification. Namely, they should provide mechanisms to analyze the complexity of workflows. High complexity in workflows may result in poor understandability, errors, defects, and exceptions leading processes to need more time to develop, test, and maintain. Therefore, excessive complexity should be avoided. The major goal of this paper is to describe a quality metric to analyze the complexity of workflow patterns from a log-based perspective.

Keywords: workflow, process log, workflow complexity, business process quality metrics, business process analysis.

1 Introduction

Workflow verification tools such as Woflan [1] are indispensable for the current generation of WfMS. Yet, another desirable category of tools that allows building better workflows are tools that implement workflow quality metrics. In the area of software engineering, quality metrics have shown their importance for good programming practices and software designs. Since there are strong similarities between software programs and business process designs, several researchers have recognized the potential of quality metrics in business process management [2-5].

In [6], Vanderfeesten et al. suggest that quality metrics to analyze business processes can be classified into four distinct categories: *coupling*, *cohesion*, *complexity*, *modularity* and *size*. In this paper we focus our attention on developing quality metrics to evaluate the complexity of workflow models [7].

Workflow complexity should not be confused with algorithmic complexity measures (e.g. Big-Oh "O"-Notation), whose aim is to compare the performance of algorithms [7]. Workflow complexity can be defined as the degree to which a workflow is difficult to analyze, understand or explain. It can be characterized by the number and intricacy of task interfaces, transitions, conditional and parallel branches, the existence of loops, roles, task categories, the types of data structures, and other workflow characteristics.

* This work was partially funded by FCT, POCTI-219, and FEDER.

In this paper, we present a metric to calculate the Log-Based Complexity (LBC) of workflow patterns [8]. Since our analysis of complexity is based on flow descriptions, we devise complexity metrics for each workflow pattern. The idea of this metric is to relate complexity with the number of different log traces that can be generated from the execution of a workflow. If a workflow always generates the same entries (i.e., the same task ID) in the process log then its complexity is minimal. On the other hand, if a workflow can generate $n!$ distinct log entries (where n is the number of tasks of a workflow) then its complexity is higher.

This paper is structured as follows. The second section presents the related work. In section 3, a new complexity metric for workflow patterns is presented. We start giving a brief overview of what workflow patterns are and explain the reasons why four patterns have not been included in our metric. In section 4, we give a practical example showing how the metric presented is to be applied to workflows. Finally, the last section presents our conclusions.

2 Related Work

The concept of process metrics has first been introduced in [7] to provide a quantitative basis for the design, development, validation, and analysis of business process models. Later the concept has been re-coined to Business Process Quality Metrics (BPQM).

The first metric presented in literature was the control-flow complexity (CFC) metric [7]. It was inspired by McCabe's cyclomatic complexity. The CFC metric was evaluated according to Weyuker's properties and an empirical study has been carried out by means of a controlled experiment [9] to validate it. In [10], Mendling proposes a density metric inspired by social network analysis in order to quantify the complexity of an EPC. In [11], the author presents a data flow complexity metric for process models. Reijers and Vanderfeesten [12] also present a metric that computes the degree of coupling and cohesion in a BOM (Bill of materials) model by analyzing data elements. Gruhn and Laue [5] use the notion of cognitive weights as a basic control structure to measure the difficulty in understanding control structures in workflows. Finally, in [6], the authors show how the ProM framework implements some of the quality metrics that have been developed so far.

3 Log-Based Complexity of Workflow Patterns

Today, many enterprise information systems store relevant events in a log. The importance of event logs makes them of value and interest to study and to evaluate the complexity of the workflows that generates them. The main idea is to compute the number of distinct logs a specific workflow can generate. The higher the number of distinct logs that can be generated, the more complex the workflow is.

To have an idea on the distinct process logs that can be generated from the execution of a workflow, let us consider the following two examples. A sequential workflow with tasks *A*, *B*, *C*, and *D* can only generate one type of process log entry. Fro example, *A12-B32-C37-D67*. But, if the workflow model defines two sequences:

1) *A* and *B*, and 2) *C* and *D*, and places these two sequences in parallel then the number of different process log entries that can be generated is 6. For example, the entries *A23-B34-C45-D56*, *A23-C45-B34-D56*, *A23-C45-D56-B34*, *C45-D-A23-B34*, *C45-A23-D56-B34*, and *C45-A23-B34-D56*. Intuitively, the second workflow is more complex from a process log perspective since it can have more "mutations". The first workflow, in our example, is predictable, while the second workflow is unpredictable. As more distinct process log entries can be generated from a workflow, the more unpredictable the workflow is considered to be.

3.1 Workflow Patterns

Aalst et al. [13] have identified a number of workflow patterns that describe the behavior of business processes and identify comprehensive workflow functionality. The advantage of these patterns lies in the ability for an in-depth comparison of a number of commercially available workflow management systems based on their capability of executing different workflow structures and their behavior. As we have discussed previously, the log-based complexity is a particular type of control-flow complexity which is influenced by elements such as splits, joins, and loops. Therefore, our first task was to identify the relevant workflow patterns for log-based complexity analysis. We concluded that all patterns, except four, were relevant for the metric we proposed to develop. The Implicit Termination, Multiple Instances without Synchronization, and Cancellation Patterns were not captured by our metric since they are implemented by a very few number of WfMS, the support can lead to an unexpected behavior, or they no not affect the log-based complexity of processes.

3.2 Log-Based Complexity Metrics for Workflow Patterns

Since it is a well known language, we have used BPMN (Business Process Modeling Notation) to illustrate the log-based complexity of workflow patterns. Of course, we could have used other languages, such as XPDL (XML Process Definition Language), or we could have taken a more formal approach using Petri nets. But we consider that BPMN is a simple and easy language to understand which facilitates readers to comprehend the number of traces introduced by a workflow pattern. To make this paper concise, we will only address a sub-set of workflow patterns. These patterns are representative and explain the rational of our approach to develop the LBC metric.

The simplest element that can generate a log entry is the execution of a task (i.e. an activity). Figure 1 illustrates the representation of a task in BPMN. Please note that the dashed line is not part of the BPMN. We use it to specify the scope of the workflow. In Figure 1, the dashed line specifies that workflow *wf* is composed of task A.

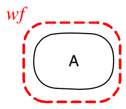


Fig. 1. A task

Since an activity only generates one entry in the process log, its log-based complexity is simply 1, i.e.:

$$LBC_T(wf) = 1$$

Sequence pattern (P1). The sequence pattern is defined as being an ordered series of tasks, with one task starting after a previous task has completed (Figure 2). Please not that BPMN graphically define a sub-workflow using a rounded box with the plus sign (+) inside.

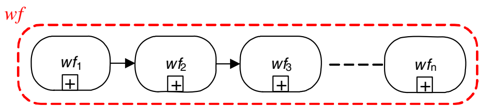


Fig. 2. The sequence pattern

The behavior of this pattern can be described by the use of a token that travels down a sequence from sub-workflow wf_1, to sub-workflow wf_2... and finally reaches sub-workflow wf_n. Since the execution of this pattern always generates the same trace in the process log, the log-based complexity of this pattern is simply given by the following formulae:

$$LBC_{P1}(wf) = \prod_{i=1}^{n} LBC_{x_i}(wf_i)$$

For example, a sequential workflow wf with two sub-workflows wf_1 and wf_2, where wf_1 can generate 4 different traces and wf_2 can generate 3 different traces has a complexity of $LBC(wf) = 4 \cdot 3 = 12$.

Exclusive Choice and Deferred Choice (P4, P16). The exclusive choice pattern (P4, XOR-split) is defined as being a location in the workflow where the flow is split into two or more exclusive alternative paths and, based on a certain condition, one of the paths is taken (Figure 3). The pattern is exclusive since only one of the alternative paths is taken. The deferred choice pattern (P16, a XOR-split abstraction) is very similar to the exclusive choice pattern. In contrast to the exclusive choice pattern, the deferred choice transition selection is based on external input while the exclusive choice relies on information being part of the workflow. Once a transition is activated, the other alternative transitions are deactivated. The moment of choice is delayed until the processing in one of the alternative transitions has actually started.

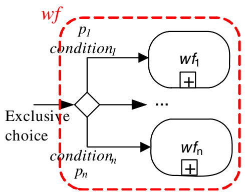


Fig. 3. The exclusive choice pattern

The behavior of these patterns can be described by the use of a token that follows only one of the outgoing transitions of the exclusive choice pattern. Since only one path of the *n* paths present can be followed, the log-based complexity is the sum of the individual complexity of each workflow $wf_1 \ldots wf_n$. Thus, the LBC for these two patterns is:

$$LBC_{P4}(wf) = LBC_{P16}(wf) = \sum_{i=1}^{n} p_i \times LBC_{x_i}(wf_i)$$

Since workflows are non-deterministic, LBC_{P_4} and $LBC_{P_{16}}$ are weighted functions, where p_i is the probability of following a specific path at runtime.

Arbitrary Cycles Loop pattern (P10). The arbitrary cycle pattern is a mechanism for allowing sections of a workflow where one or more activities can be done repeatedly (i.e. a loop). Figure 4 shows an example of the use of the arbitrary cycle pattern.

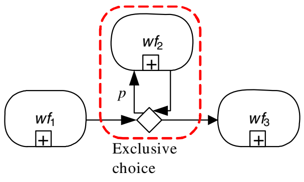


Fig. 4. The arbitrary cycle pattern

At runtime, one of the following scenarios can occur:

wf_1-wf_3	$P_0=1-p$
wf_1-wf_2-wf_3	$P_1=p(1-p) * 1 * LBC_x(wf_2)$
wf_1-wf_2-wf_2-wf_3	$P_2=p^2(1-p) * 2 * LBC_x(wf_2)$
wf_1-wf_2-wf_2-wf_2-wf_3	$P_3=p^3(1-p) * 3 * LBC_x(wf_2)$
…	
wf_1-wf_2-…-wf_2-wf_3	$P_{L-1}=p^{L-1}(1-p) * (L-1) * LBC_x(wf_2)$
wf_1-wf_2-…-wf_2-wf_3	$P_L=(p^L(1-p)+p^{L+1}) * L * LBC_x(wf_2)$

The variable P_j (for $0 \le j \le L$, L=maximum number of iterations) indicates the probability of a specific case to occur at runtime when the probabilities of repeating and escaping the loop are p and $(1-p)$, respectively, in every iteration ($0<p<1$). It is assumed to force a compulsorily escape from the loop after L iterations (the probability of such a case is p^{L+1}). Therefore, we can calculate the log-based complexity of the loop as follows:

$$LBC_{P10}(wf) = \left(\sum_{j=0}^{L-1} p^j (1-p) \times j \times LBC_x(wf_2) \right) + (p^L(1-p) + p^{L+1}) \times L \times LBC_x(wf_2)$$

Interleaved parallel routing pattern (P17). In this pattern, a set of activities is executed with no specific order. The performers of the activities will decide the order of the activities. Each task in the set is executed and no two activities are executed at the same moment. It is not until one task is completed that the decision on what to do next is taken.

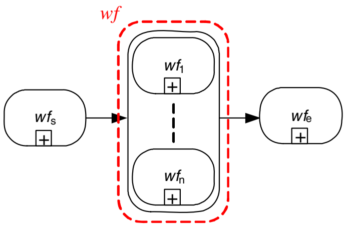


Fig. 5. The interleaved parallel routing pattern

Figure 5 illustrates the interleaved parallel routing pattern. Once sub-workflow wf_s is completed, a token is transferred to the set of sub-workflow wf_1, …, wf_n. The token will be assigned to one of the sub-worklfows wf_1, wf_2,…, or wf_n and then transferred to another sub-workflow until all the sub-workflows are completed. This is done sequentially. Since all sub-workflows will be activated at some point in time in any order, we have $n!$ permutations for the sub-workflows, therefore the log-based complexity is:

$$LBC_{P17}(wf) = n! \times \prod_{i=1}^{n} LBC_{x_i}(wf_i)$$

4 Aggregating the Complexity of Workflow Patterns

Having devised custom metrics for each workflow pattern, we can calculate the LBC of workflows. Our approach to calculate the overall log-based complexity of a workflow consists in the stepwise collapsing of the workflow into a single node by alternately aggregating workflow patterns. The algorithm that we use repeatedly applies a set of workflow transformation rules (based on the workflow patterns that we have analyzed) to a workflow until only one atomic task remains. Each time a transformation rule is applied, the workflow structure changes. After several iterations only one task will remain. When this state is reached, the remaining task contains the complexity corresponding to the initial workflow under analysis.

Figure 6 illustrates the set of transformation rules that are applied to an initial workflow to compute the log-based complexity. To the initial process, illustrated in Figure 12.a), we apply patterns LBC_T and LBC_{P13}. The resulting process is illustrated in Figure 12.b). To this new process we apply patterns LBC_T, LBC_{P1}, LBC_{P5}, and LBC_{P13}. The process suffers various transformations as shown in Figures 12.c) and Figure 12.d). Finally, after the last transformation, only one task remains (Figure 12.e) and this task (ABCDEnEF) contains the overall complexity of the workflow which is 5.75. This indicates that the initial workflow can generates, on average (since the workflow is non-deterministic) 5.75 distinct process logs.

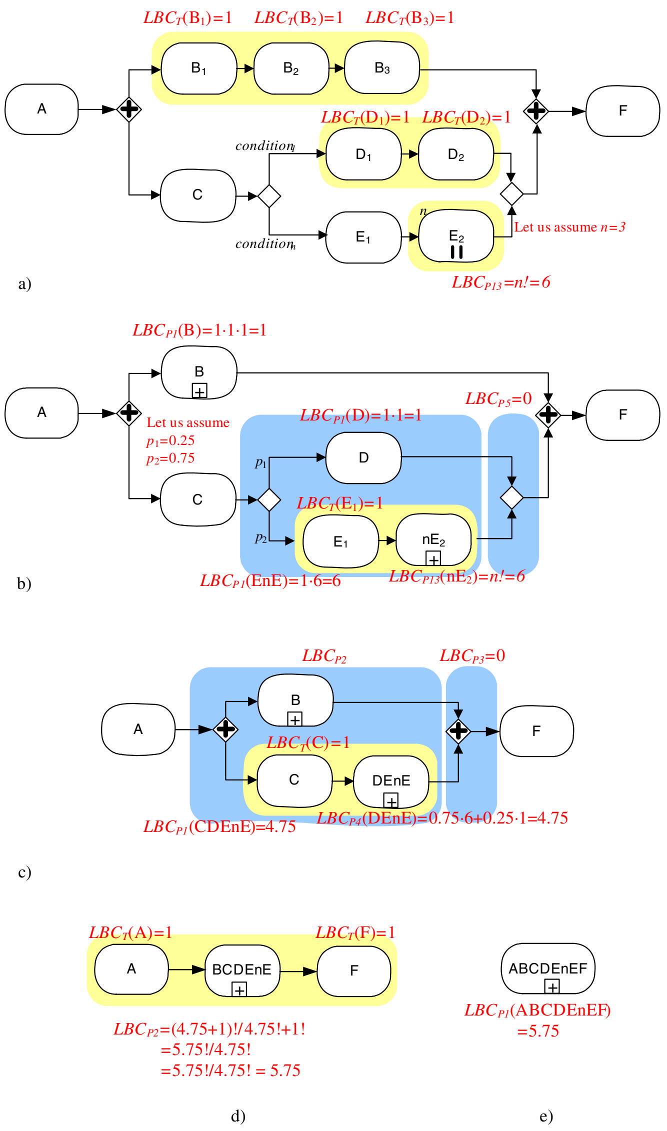


Fig. 6. Log-based complexity computation

5 Conclusions

Recently, a new approach to workflow analysis has been proposed and targets the development of Business Process Quality Metrics (BPQM) to evaluate workflow models. One particular class of quality metrics has the goal of analyzing the complexity of workflow models. This analysis enables to identify complex workflows that require reparative actions to improve their comprehensibility. To enlarge the number of approaches available to analyze workflows, in this paper, we presented the log-based complexity (LBC) metric to calculate the complexity of workflows. Our approach consisted of devising a complexity metric based on the number of process logs that are generated when workflows are executed. Our complexity metric is a design-time measurement and can be used to evaluate the difficulty of producing a workflow design before its implementation.

References

1. Verbeek, H.M.W., Basten, T., Aalst, W.M.V.d.: Diagnosing workflow processes using woflan. The Computer Journal 44(4), 246–279 (2001)
2. Gruhn, V., Laue, R.: Adopting the Cognitive Complexity Measure for Business Process Models. In: 5th IEEE International Conference on Cognitive Informatics, Beijing, China, IEEE Computer Society, IEEE Computer Society (2006)
3. Latva-Koivisto, A.M.: Finding a complexity measure for business process models, Helsinki University of Technology, Systems Analysis Laboratory: Helsinki (2001)
4. Cardoso, J., et al.: A Discourse on Complexity of Process Models. In: BPI 2006. Second International Workshop on Business Process Intelligence, In conjunction with BPM 2006, Vienna, Austria, Springer, Heidelberg (2006)
5. Gruhn, V., Laue, R.: Complexity Metrics for Business Process Models. In: 9th International Conference on Business Information Systems, Klagenfurt, Austria: GI (2006)
6. Vanderfeesten, I., et al.: Quality Metrics for Business Process Models. In: Fischer, L. (ed.) Workflow Handbook 2007, pp. 179–190. Future Strategies Inc., Lighthouse Point, FL, USA (2007)
7. Cardoso, J.: Evaluating Workflows and Web Process Complexity. In: Fischer, L. (ed.) Workflow Handbook 2005, pp. 284–290. Future Strategies Inc., Lighthouse Point, FL, USA (2005)
8. Aalst, W.M.P.v.d., et al.: Workflow Patterns. Distributed and Parallel Databases 14(3), 5–51 (2003)
9. Cardoso, J.: Process control-flow complexity metric: An empirical validation. In: IEEE SCC 2006. IEEE International Conference on Services Computing, Chicago, USA, IEEE Computer Society, Los Alamitos (2006)
10. Mendling, J.: Testing Density as a Complexity Metric for EPCs, Technical Report JM-2006, 11-15. 2006, Vienna University of Economics and Business Administration, Austria (2006)
11. Cardoso, J.: About the Data-Flow Complexity of Web Processes. In: 6th International Workshop on Business Process Modeling, Development, and Support: Business Processes and Support Systems: Design for Flexibility, Porto, Portugal (2005)
12. Reijers, H.A., Vanderfeesten, I.T.P.: Cohesion and Coupling Metrics for Workflow Process Design. In: Desel, J., Pernici, B., Weske, M. (eds.) BPM 2004. LNCS, vol. 3080, pp. 290–305. Springer, Heidelberg (2004)
13. Aalst, W.M.P.v.d., et al.: Advanced Workflow Patterns. In: Seventh IFCIS International Conference on Cooperative Information Systems, Eilat, Israel (2000)

Distributed Objects and Applications (DOA) 2007 International Conference

DOA 2007 PC Co-chairs' Message

Welcome to the Ninth International Symposium on Distributed Objects, Middleware and Applications (DOA 2007), held in Vilamoura, Portugal, November 25–30, 2007.

The DOA conferences provide a forum for exchanging the latest research results on distributed objects, components, services, middleware and applications. To emphasize the increasing importance and proliferation of higher-level software abstractions and the associated general-purpose middleware, the term 'middleware' was added to the title of DOA this year. Research in objects, middleware and their application establishes new principles that open the way to solutions that can meet the requirements of tomorrow's applications. Conversely, practical experience in real-world projects drives this same research by exposing new ideas and posing new types of problems to be solved. With DOA 2007 we explicitly intended to provide a forum to help this mutual interaction occur, and to trigger and foster it. Submissions were therefore welcomed along both these dimensions: research (fundamentals, concepts, principles, evaluations, patterns, and algorithms) and practice (applications, experience, case studies, and lessons). Contributions attempting to cross over the gap between these two dimensions were particularly encouraged. Toward this goal, we accepted both research and experience papers.

The resulting high-quality program would not have been possible without the authors who chose DOA as a venue for their publications. All papers were submitted to a rigorous reviewing process with three reviews per paper, and considerable discussion took place among the Program Committee before decisions were taken. Out of 52 submitted papers, we finally selected 20 full papers and 4 posters. Our congratulations and thanks to the successful authors, who came from many parts around the globe to present their work in Vilamoura.

Rounding up this excellent program, Mark Little, Director of Engineering at Red Hat, was our keynote speaker discussing "Transaction Processing in a Service Oriented Architecture."

We are grateful for the dedicated work of the experts in the field from all over the world who served on the Program Committee, and whose names appear in this volume. Special thanks go to the external referees who volunteered their time to provide additional reviews. Finally, we are indebted to Kwong Yuen Lai, who was immensely helpful in facilitating the review process and making sure that everything stayed on track.

August 2007

Pascal Felber
Calton Pu
Aad van Moorsel

WS-CAF: Contexts, Coordination and Transactions for Web Services

Mark Little

Technical Development Manager, Red Hat

Abstract. As Web services have evolved as a means to integrate processes and applications at an inter-enterprise level, traditional transaction semantics and protocols have proven to be inappropriate. Web services-based transactions, colloquially termed Business Transactions, differ from traditional transactions in that they execute over long periods, they require commitments to the transaction to be "negotiated" at runtime, and isolation levels have to be relaxed. A solution to this problem has to work over HTTP and include existing transaction processing technologies of all types: database management systems, application servers, message queuing systems and packaged applications. In this paper we'll look at the WS-CAF standardization effort and show how it is attempting to address this important and difficult subject. We'll also consider how the architecture defined by WS-CAF fits into the evolving architecture of Web services

1 Introduction

It is often said that Web services are immature and missing some features compared to other distributed computing development environments such as CORBA, DCOM, and J2EE, but exactly what needs to be added is the subject of considerable debate. Many proposals have surfaced in the form of Web services specification drafts and as the number of proposals and specifications grows, confusion often grows rather than shrinks.

However, a common foundation exists underneath many of these missing pieces: *context management* (essentially the ability to associate disparate entities within the same unit of distributed work). This fact, and the problem that no such facility exists in Web services, came to light soon after SOAP 1.1 was submitted to W3C, when we first started work on mapping the Transaction Internet Protocol (TIP) to SOAP back in mid-2000. Transactioning is often mentioned as one of the major features for distributed computing environments and CORBA, DCOM/.NET, and J2EE all provide it, so it is a fairly obvious requirement for Web services. However, we quickly realized we had a larger problem than simply adding a transaction context to the SOAP header and we had to suspend the effort.

TIP was dependent on a *session-oriented communication protocol* for exchanging two-phase commit commands. Like most distributed transaction protocols, TIP required a persistent transaction context to be shared among the communicating parties in the transactional operation so that a two-phase commit protocol can be executed reliably. An abort (or rollback) can be triggered automatically when a communication connection is dropped. It is too risky to the health of the resource managers being coordinated not to rollback when communication is lost, but without a persistent session mechanism, the client (the transaction root) is unable to detect connection loss.

Unfortunately this type of behavior is in fact impossible to define for a communication system based on HTTP, where sessions are maintained only long enough to transfer an HTML page and are dropped immediately afterward. This behavior is tremendously helpful to support a system of the scale of the World Wide Web, but it is not so helpful when you need to support a classic transactioning protocol such as two-phase commit.

Many other typical features and functions of distributed systems also depend upon persistent sessions, including secure sessions, conversations, and load balancing and failover mechanisms. The way to think about the use of what we are calling *persistent sessions* in general is the ability to get back to the same place where you left off in a remote program on a subsequent call. In the specific cases of transactions, or secure conversations, for example, this is the ability to maintain the context of the first operation while waiting for the next to arrive.

As we shall see in the following sections, the OASIS Web Services Composite Application Framework [2] attempts to solve this problem by defining core support in the Web services architecture for context management. It also builds upon this to provide the necessary transaction functionality that we were unable to accomplish back in 2000.

2 An Overview of WS-CAF

In general, *composite applications* are increasing in importance as companies combine off-the-shelf and homegrown Web services into new applications. Various mechanisms are being proposed and delivered to market daily to help improve this process. New “fourth generation” language developments tools are emerging that are specifically designed to stitch together Web services from any source, regardless of the underlying implementation.

A large number of vendors are starting to sell business process management, workflow and orchestration tools for use in combining Web services into automatic business process execution flows. In addition, a growing number of businesses find themselves creating new applications by combining their own Web services with Web services available from the Internet supplied by the likes of Amazon.com and Google.com.

These types of composite applications represent a variety of requirements, from needing a simple way to share persistent data to the ability to manage recovery scenarios that include various types of transactional software. Composite applications therefore represent a significant challenge for Web services standards since they are intended to handle complex, potentially long-running interactions among multiple Web services as well as simple and short-lived interactions.

The WS-CAF suite includes three specifications that can be implemented incrementally to address the range of requirements needed to support a variety of simple to complex composite applications:

- Web Service Context (WS-CTX), a lightweight framework for simple context management.
- Web Service Coordination Framework (WS-CF), which defines the behavior of a coordinator with which Web services can register for context augmentation and results propagation, and on top of which can be plugged various transaction protocols.
- Web Services Transaction Management (WS-TXM), comprising three distinct protocols for interoperability across multiple transaction managers and supporting multiple transaction models (two phase commit, long running actions or compensation, and business process flows).

The overall aim of the combination of the parts of WS-CAF is to provide a complete solution that supports various transaction processing models and architectures. Implementations of WS-CAF can start small and grow to include more functionality over time. WS-CAF specifications are designed to compliment Web services orchestration and choreography technologies such as WS-BPEL [3] and WSCI [4] and are compatible with other Web services specifications. The emphasis of WS-CAF is to define supporting services required by Web services used in combination, including other specifications.

The parts of WS-CAF comprise a stack, starting from WS-CTX, adding WS-CF, and finally WS-TXM to deliver the complete features and functionality required by composite applications. An implementation of WS-CAF can start with WS-CTX for simple context management, and later add WS-CF for its additional context management features and context message delivery guarantees, and finally add WS-TXM for managing a variety of recovery protocols.

In the following sections we shall examine each of these specifications in more detail and show how they support the development of composite Web Service applications.

2.1 Context Management

The ability to scope units of work (known as activities) is a requirement in a variety of aspects of distributed applications. In order to correlate the work of multiple Web Services within the same activity, it is necessary to propagate additional information called the context to each participating service. The context contains information such as a unique ID that allows a series of operations to share a common outcome, and is propagated in a SOAP header block whenever application messages are transmitted between component services. The reliable management of the contexts that provide distributed application scope is addressed by the WS-Context specification.

The purpose of a context is to allow multiple individual Web Services to enter a relationship by sharing certain common attributes as an externally modeled entity. Typical reasons for Web Services to share context include common security domains where multiple Web Services execute within the scope of a single authorized session, or common outcome negotiation where each party within the activity needs to know whether or not each of the other participants successfully completed their work.

Through the use of shared context Web services from different sources can effectively become part of the same application because they share common system information. A classic example is a single sign-on mechanism that allows a user or an application to present authentication credentials to access to a set of cooperating Web services. Application level context, such as a shared document, can also benefit from a generic context management service.

In general terms, a context defines basic information about the activity structure. The context contains information necessary for multiple Web Services to be associated with the same activity that may be dynamically updated by services as the application or process makes progress.

WS-Context defines a context data structure that can be arbitrarily augmented. By default, all the context defines is a unique context identifier, the type of the context (e.g., transaction or security) and a timeout value (how long the context can remain valid). Like SOAP headers, which WS-Context can replace or combine for easier management, the context data structure includes an attribute requiring the context to be understood and/or propagated. For example:

```
<ContextType> MyContext </ContextType>
  <context-identifier>
   www.webservicestransactions.org/example/ContextExample
  </context-identifier>
. . .
. . . mustUnderstand=true
. . . mustPropagate=true
. . .
  <child-contexts>
    <child-context>
      <user-name> HomerSimpson </user-name>
      <password> ******** </password>
    </child-context>
    <child-context>
      <database-name> SQL-DB </database-name>
      <file-name> Index-S-file </file-name>
      <display-address> PocketPc25 </display-address>
    </child-context>
    <child-context>
      <transaction-type> BusinessProcess </transaction-type>
      <transaction-mode> Required </transaction-mode>
    </child-context>
  </child-contexts>
```

The context structure shown above includes "children" that can be used to share information needed to process a request on behalf of the user of a composite Web service. In this case, the context includes the `mustUnderstand` attribute set to `true` to indicate that the context must be understood in order to process the request, since it contains information necessary for successful completion of the request. The context has also been marked as `mustPropagate=true`, meaning that each Web service in the composite must receive or be able to access the context to ensure proper execution.

The example illustrates user information that obtains a security token and passes the token as a single sign-on feature for the composite application. In other words, the context could be provided as input to the first Web service in a WS-BPEL defined flow. The first Web service in the flow then could check the username and password (the asterisks are used to indicate opaque data in the example) and retrieve an authentication token to use in checking whether the user is authorized to access each subsequent Web service in the flow. Such an authentication token would be placed back into the context data structure as an augmentation to the original structure. For example:

```
<child-context>
  <user-name> HomerSimpson </user-name>
  <password> ******** </password>
  <AuthToken> ******** </AuthToken>
</child-context>
```

The `AuthToken` is added by the security system at the end of the username and password information upon execution of the initial Web service in the flow. The context is a living data structure; the results of a security sign on (or other operation pertinent to the contents of the context) would typically be added for propagation to the next Web service in the flow. For example, a single sign on system bridging multiple security domains would add another token to the context.

Web Services sessions

It has long been recognized that the World Wide Web is probably the most successful distributed system created. It is inherently loosely coupled (clients and servers frequently interact across the globe) and highly scaleable (many thousands of Web sites). There are a number of factors that can be attributed to the Web's success, but two of the most important are:

- Sessions between clients and servers are maintained only long enough to transfer an HTML page and are dropped immediately afterward. This means that costly resources (e.g., TCP/IP connections, threads, processes) are not maintained for long durations, particularly when there are many users interacting with a service.
- Server interactions are either stateless, meaning that any instance of a Web server offering a particular service, e.g., airline reservation, can field the request, or information required to identify a previous user (and possibly state) is propagated with the invocation, e.g., the cookie.

Both of these factors mean that clusters of servers can relatively easily be used to distribute the load and provide improved availability/fault-tolerance to users. Web servers offering critical services are typically deployed over a cluster of machines. A locally distributed cluster of machines with the illusion of a single IP address and capable of working together to host a Web site provides a practical way of scaling up processing power and sharing load at a given site. Commercially available server clusters rely on a specially designed gateway router to distribute the load using a mechanism known as network address translation (NAT). The mechanism operates by editing the IP headers of packets so as to change the destination address before the IP to host address translation is performed. Similarly, return packets are edited to change their source IP address. Such translations can be performed on a per session basis so that all IP packets corresponding to a particular session are consistently redirected.

Most proponents of Web Services agree that it is important that its architecture is as scalable and flexible as the Web. As a result, the current interaction pattern for Web Services is based on coarse-grained services or components. The architecture is deliberately not prescriptive about what happens behind service endpoints: Web Services are ultimately only concerned with the transfer of structured data between parties, plus any meta-level information to safeguard such transfers (e.g., by encrypting or digitally signing messages). This gives flexibility of implementation, allowing systems to adapt to changes in requirements, technology etc. without directly affecting users. Furthermore, most businesses will not want to expose their back-end implementation decisions and strategies to users for a variety of reasons.

In distributed systems such as CORBA, J2EE and DCOM, interactions are typically between stateful objects that resided within *containers*. In these architectures, objects are exposed as individually referenceable entities, tied to specific containers and therefore often to specific machines. Because most Web Services applications are written using object-oriented languages, it is natural to think about extending that architecture to Web Services. Therefore a service exposes *Web Services resources* that represent specific states. The result is that such architectures produce tight coupling between clients and services, making it difficult for them to scale to the level of the World Wide Web.

Right now, there are two primary models for the session concept that are being defined by companies participating in defining Web services: the WS-Addressing EndpointReference with ReferenceProperties [12] and the WS-Context explicit context structure. The WS-Addressing session model provides coupling between the web service endpoint information and the session data, which is analogous to object references in distributed object systems. WS-Context provides a session model that is an evolution of the session models found in HTTP servers, transaction, and MOM systems, allowing a service client to more naturally bind the relationship to the service dynamically and temporarily [11]. The client's communication channel to the service is not impacted by a specific session relationship.

If a session-like model based on WS-Addressing were to be used when interacting with stateful services, then the tight coupling between state and service would impact on clients. As in other distribution environments where this model is used (e.g., CORBA or J2EE), the remote reference (address) that the client has to the service endpoint *must* be remembered by the client for subsequent invocations. If the client application interacts with multiple services within the same logical session, then it is often the case that the state of a service has relevance to the client only when used in conjunction with the associated states of the other services. This necessarily means that the client must remember each service reference and somehow associate them with a specific interaction; multiple interactions will obviously result in different reference sets that may be combined to represent each sessions.

For example, if there are N services used within the same application session, each maintaining m different states, the client application will have to maintain N*m reference endpoints. It is worth remembering that the initial service endpoint references will often be obtained from some bootstrap process such as UDDI. But in this model, these references are stateless and of no use beyond starting the application interactions. Subsequent visits to these sites that require access to specific states must use different references in the WS-Addressing model.

This obviously does not scale to an environment the size of the Web. However, an alternative approach is to use WS-Context and continue to embrace the inherently loosely coupled nature of Web Services. As we have shown, each interaction with a set of services can be modeled as a session, and this in turn can be modeled as a WS-Context activity with an associated context. Whenever a client application interacts with a set of services within the same session, the context is propagated to the services and they map this context to the necessary states that the client interaction requires.

How this mapping occurs is an implementation specific choice that need not be exposed to the client. Furthermore, since each service within a specific session gets the same context, upon later revisiting these services and providing the same context again, the client application can be sure to return to a consistent set of states. So for the N services and m states in our previous example, the client need only maintain N endpoint references and as we mentioned earlier, typically these will be obtained from the bootstrap process anyway. Thus, this model scales much better.

2.2 Coordination

A *coordinator* is a software entity responsible for ensuring consensus is achieved between multiple parties. Coordinators exist in CORBA, .NET, J2EE, and other distributed computing environments to coordinate the classic two-phase commit transaction protocol across multiple data resources. However, coordination is a more fundamental requirement: it is used in security, replication, caching and other areas.

Therefore, the definition of a coordinator in WS-CAF is extended for use with Web services by using a plug in mechanism that supports multiple coordination protocols such as the classic two-phase commit protocol, long running actions with compensation, and complex business process and orchestration flows.

Web services are designed to be multi-protocol and therefore to map to multiple underlying technologies. Instead of tying the coordinator to the two-phase commit protocol, which is the way current coordinators are defined, the WS-CF specification creates a general-purpose coordinator capable of driving a variety of context types and transaction protocols (such as those defined in WS-TXM and others).

```
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
  <env:Header>
    <n:Composite xmlns:n=”http://example.org/CompositeApplication”>
      <n:Coordinator>
       http://www.webservicestransactions.org/example/CoordinatorURI
      </Cooordinator>
    </n:Composite>
  </env:Header>
  <env:Body> ...
</env:Envelope>
```

In the above example, the coordinator URI points to a Web service interface that defines the SOAP message pattern for interactions between the coordinator and the Web service execution. The coordinator then manages any user defined context and generates and propagates any context for use within the operations of the composite and includes each registered Web service in the recovery protocol. When multiple Web services register with the coordinator to use the same context type, the message exchange pattern includes all Web service executions within the composite. In other words, the scope for a given context type is determined by the Web services that register with the coordinator to share it.

The message exchange pattern described for the Web services in the application isn't changed. By registering with the coordinator, however, a separate message exchange pattern is established as a secondary, system-level interaction to handle the context propagation and recovery operations. The two message exchange patterns are linked using the context ID passed in the SOAP header and given to the coordinator upon registration.

2.3 Transactions

Distributed systems pose reliability problems not frequently encountered in centralized systems. A distributed system consisting of a number of computers connected by a network can be subject to independent failure of any of its components, such as the computers themselves, network links, operating systems, or individual applications. Decentralization allows parts of the system to fail while other parts remain functioning, which leads to the possibility of abnormal behavior of executing applications.

Consider the case of a distributed system where the individual computers provide a selection of useful services, which can be utilized by an application. It is natural that an application that uses a collection of these services requires that they behave consistently, even in the presence of failures. A very simple consistency requirement is that of *failure atomicity*: the application either terminates normally, producing the intended results, or is aborted, producing no results at all. This failure atomicity property is supported by Atomic transactions, which have the following familiar *ACID* properties:

- *Atomicity*: The transaction completes successfully (commits) or if it fails (aborts) all of its effects are undone (rolled back);
- *Consistency*: Transactions produce consistent results and preserve application specific invariants;
- *Isolation*: Intermediate states produced while a transaction is executing are not visible to other transactions. Furthermore transactions appear to execute serially, even if they are actually executed concurrently. This is typically achieved by locking resources for the duration of the transaction so that they cannot be acquired in a conflicting manner by another transaction;
- *Durability*: The effects of a committed transaction are never lost (except by a catastrophic failure).

A transaction can be terminated in two ways: *committed* or *aborted* (rolled back). When a transaction is committed, all changes made within it are made durable (forced on to stable storage such as disk). When a transaction is aborted, all changes made during the lifetime of the transaction are undone. In addition it is possible to nest atomic transactions; where the effects of a nested action are provisional upon the commit/abort of the outermost (top-level) atomic transaction.

Why ACID transactions may be too Strong

Traditional transaction processing systems are sufficient to meet requirements if an application function can be represented as a single top-level transaction. However, this is frequently not the case. Top-level transactions are most suitably viewed as short-lived entities, performing stable state changes to the system; they are less well suited for structuring long-lived application functions that run for minutes, hours, days, or longer. Long-lived top-level transactions may reduce the concurrency in the system to an unacceptable level by holding on to resources (usually by locking) for a long time. Furthermore, if such a transaction aborts much valuable work already performed will be undone.

Given that the industry is moving towards loosely coupled, coarse-grained B2B interaction model supported by Web services, it has become clear that the semantics of traditional ACID transactions are unsuitable for Web scale deployment. Web services-based transactions differ from traditional transactions in that they execute over long periods, they require commitments to the transaction to be negotiated at runtime, and isolation levels have to be relaxed.

The WS-TXM Approach

Given that we have already seen that traditional transaction models are not appropriate for Web services, we must pose the question, "what type of model or protocol *is* appropriate?" The answer to that question is that that no one specific protocol is likely to be sufficient, given the wide range of situations that Web service transactions are likely to be deployed within. Hence the WS-TXM specification proposes three distinct models, which support the semantics of a particular kind of B2B interaction.

WS-ACID

An atomic transaction or WS-ACID is similar to traditional ACID transactions and intended to support short-duration interactions where ACID semantics are appropriate.

Within the scope of a WS-ACID transaction, services typically enroll transaction-aware resources, such as databases and message queues, indirectly as participants under the control of the transaction. When the transaction terminates, the outcome decision of the WS-ACID transaction is then propagated to each enlisted resource via the participant, and each takes the appropriate commit or rollback actions.

This protocol is very similar to those employed by traditional transaction systems that already form the backbone of an enterprise. It is assumed that all services (and associated participants) provide ACID semantics and that any use of atomic transactions occurs in environments and situations where this is appropriate: in a trusted domain, over short durations.

Long Running Activities
The long running action model (LRA) is designed specifically for those business interactions that occur over a long duration. Within this model, an activity reflects business interactions: all work performed within the scope of an application is required to be compensatable. Therefore, an application's work is either performed successfully or undone. How individual Web services perform their work and ensure it can be undone if compensation is required, are implementation choices and not exposed to the LRA model. The LRA model simply defines the triggers for compensation actions and the conditions under which those triggers are executed.

There is a caveat to this model though, where application services may not be compensatable (e.g. an application-level service that prints and mails cheques), or the ability to compensate may be transient. The LRA model allows applications to combine services that can be compensated with those that cannot be compensated. Obviously by mixing the two service types the user may end up with a business activity that will ultimately not be undone by the LRA model, but which may require outside (application specific) compensation.

The LRA model defines a protocol actor called a Compensator that operates on behalf of a service to undo the work it performs within the scope of an LRA. How compensation is carried out will obviously be dependant upon the service; compensation work may be carried out by other LRAs which themselves have Compensators.

When a service performs work that may have to be later compensated within the scope of an LRA, it enlists a Compensator participant with the LRA coordinator. The coordinator will send the Compensator one of the following messages when the activity terminates:

- *Success*: the activity has completed successfully. If the activity is nested then compensators may propagate that outcome to the enclosing LRA.
- *Fail*: the activity has completed unsuccessfully. All compensators that are registered with the LRA will be invoked to perform compensation in reverse order. The coordinator forgets about all compensators that indicated they operated correctly. Otherwise, compensation may be attempted again or alternatively a compensation violation has occurred and must be logged.

LRAs may be used both sequentially and concurrently, where the termination of an LRA signals the start of some other unit of work within an application. However, LRAs are units of compensatable work and an application may have as many such units of work operating simultaneously as it needs to accomplish its tasks. Furthermore, the outcome of work within LRAs may determine how other LRAs are terminated.

An application can be structured so that LRAs are used to assemble units of compensatable work and then held in the active state while the application performs other work in the scope of different (concurrent or sequential) LRAs. Only when the right subset of work (LRAs) is arrived at by the application will that subset be confirmed; all other LRAs will be told to cancel (complete in a failure state).

As we have seen, in the LRA model each application is bound to the scope of a compensation interaction. For example, when a user reserves a seat on a flight, the airline reservation centre may take an optimistic approach and actually book the seat and debit the users account, relying on the fact that most of their customers who reserve seats later book them; the compensation action for this activity would obviously be to un-book the seat and credit the user's account. Work performed within the scope of a nested LRA must remain compensatable until an enclosing service informs the individual service(s) that it is no longer required.

Let's consider another example of a long running business transaction. The application is concerned with booking a taxi, reserving a table at a restaurant, reserving a seat at the theatre, and then booking a room at a hotel. If all of these operations were performed as a single transaction then resources acquired during booking the taxi (for example) would not be released until the top-level transaction has terminated. If subsequent activities do not require those resources, then they will be needlessly unavailable to other clients.

Figure 1 shows how part of the night-out may be mapped into LRAs. All of the individual activities are compensatable. For example, this means that if LRA1 fails or the user decides to not accept the booked taxi, the work will be undone automatically. Because LRA1 is nested within another LRA, once LRA1 completes successfully any compensation mechanisms for its work may be passed to LRA5: this is an implementation choice for the Compensator. In the event that LRA5 completes successfully, no work is required to be compensated, otherwise all work performed within the scope of LRA5 (LRA1 to LRA4) will be compensated.

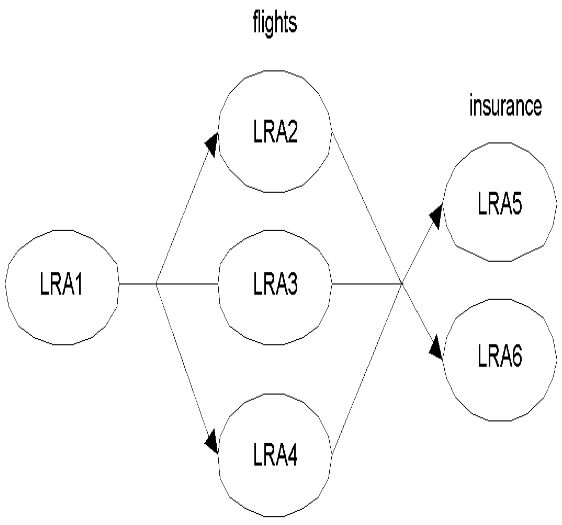


Fig. 1. LRA example

Business Process

The Business Process (BP) protocol is significantly different from any of the other transaction models we have seen to-date (and there is no directly comparable model in Web Services transactions specifications). This model is specifically aimed at tying together heterogeneous transaction domains into a single business-to-business transaction. So, for example, with the BP model it is possible to have a long-running business transaction span messaging, workflow and traditional ACID transactions, allowing enterprises to leverage their existing IT investment.

In the business process transaction model (BP model) all parties involved in a business process reside within *business domains*, which may themselves use business processes to perform work. Business process transactions are responsible for managing interactions *between* these domains. A business process (business-to-business interaction) is split into *business tasks* and each task executes within a specific business domain. A business domain may itself be subdivided into other business domains (business processes) in a recursive manner.

Each domain may represent a different transaction model if such a federation of models is more appropriate to the activity. Each business task (which may be modeled as a scope) may provide implementation specific counter-effects in the event that the enclosing scope must cancel. In addition, periodically the controlling application may request that all of the business domains checkpoint their state such that they can either be consistently rolled back to that checkpoint by the application, or restarted from the checkpoint in the event of a failure.

An individual task may require multiple services to work. Each task is assumed to be a compensatable unit of work. However, as with the LRA model described earlier, how compensation is provided is an implementation choice for the task.

For example, consider the purchasing of a home entertainment system example shown in Figure 2. The on-line shop interacts with its specific suppliers, each of which resides in its own business domain. The work necessary to obtain each component is modeled as a separate task, or Web service. In this example, the HiFi task is actually composed of two sub-tasks.

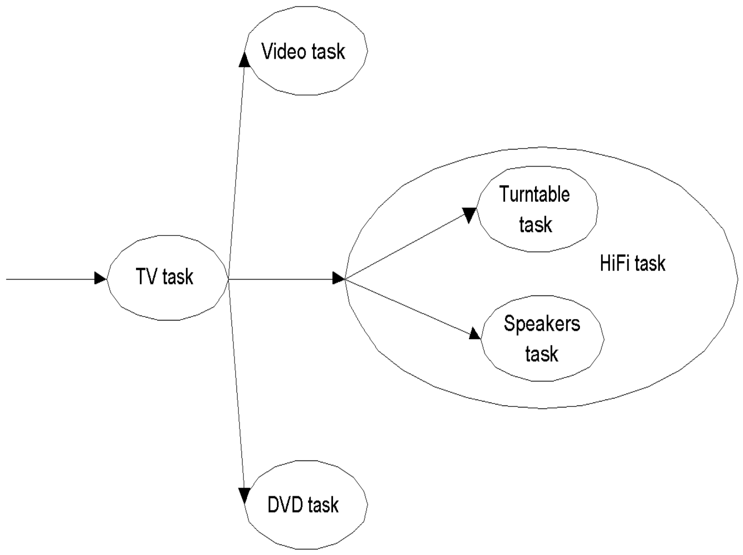


Fig. 2. Business processes and tasks

In this example, the user may interact synchronously with the travel agent to build up the details of the holiday required. Alternatively, the user may submit an order (possibly with a list of alternate requirements, such as destinations, dates, etc.) to the agent who will eventually call back when it has been filled; likewise, the travel agent then submits orders to each supplier, requiring them to call back when each component is available (or is known to be unavailable).

Business domains are instructed to perform work within the scope of a global business process. The business process has an overall manager that may be informed by individual tasks when they have completed their work or it may periodically communicate with each task to determine its current status. In addition, each task may make checkpoints of its progress such that if a failure occurs, it may be restarted from that point rather than having to start from the beginning. A business process can either terminate in a confirmed (successful) manner in which case *all* of the work requested will have been performed, or it will terminate in a cancelled (unsuccessful) manner, in which case all of the work will be undone.

If it cannot be undone, then this fact must be logged.

3 Comparison with Other Specifications

The WS-CAF specifications are designed to work with and complement other Web services specifications, including WS-Security, WS-Reliability, WS-BPEL, and others. The WS-CAF specifications define the SOAP message exchange patterns and WSDL interfaces necessary to accomplish the context management, coordination, and transaction processing capabilities needed to support composite application executions.

The question of compatibility with other Web services specifications is a difficult one since so many specifications are under progression at various standards bodies and through private consortia. It's often hard to know where any particular Web services specification fits within the overall picture. The W3C is producing a Web Services Architecture specification on this topic [5], while IBM and Microsoft have produced a whitepaper to reflect their own view [6]. At this point in time neither seems definitive, which is understandable given the rate of change still occurring in Web services and the fact that no single standards body is in control, and that so many specifications remain under private copyright.

An important consideration with respect to Web services specifications is the issue of intellectual property rights and copyright ownership. The WS-Interoperability organization [7] for example has debated to what extent their profiles can or should reference private specifications. The WS-I Basic Profile references SOAP 1.1, WSDL 1.1, and UDDI V2, all of which were produced by private consortia but have since been submitted to a standards body.

Some specifications under private copyright ownership require royalty fees to be paid to the copyright owners for the right to implement and sell software based upon them. Web services vendors who are not copyright holders on a given specification may be concerned about implementing a specification that their competitors control, especially when they are not allowed to participate in its definition or evolution.

When a specification is not under the control of a single vendor of group of vendors, it's said to be "open," meaning that anyone can participate in its definition and evolution.

With respect to other Web services specifications, both private and open, the WS-Context specification is unique. No other specification exists that defines a generic context management mechanism for Web services.

The OASIS WS-Coordination Framework specification shares a common derivation with the OASIS WS-Coordination specification – both are based on the Object Management Group's (OMG) extended transaction specification called *Additional Structuring Mechanisms for the OTS* [8]. This specification was developed as an extension of the Object Transaction Specification (OTS) [9], which defines how coordination works for both the CORBA and J2EE worlds.

The *Additional Structuring Mechanisms* specification, sometimes called the *Activity Specification* since it defines generic activities, pioneered the concept of a pluggable coordinator. The specification includes an example of an open nested transaction model to validate the design of a coordinator as a generic state machine capable of supporting multiple transaction protocols, rather than tying the coordinator to the two-phase commit protocol (as it is in the base OTS specification). WS-CF, like WS-C, is derived from this pioneering OMG work.

The base OTS specification also contains a precedent for WS-CAF because it defines how multiple coordinators can work together. The concept is called *interposition*, and it means that a coordinator can act as a resource to another coordinator on behalf of a set of local resources. The idea was included in the OTS specification as a network optimization, but it turns out to be useful for interoperability as well.

In IONA's Orbix Mainframe product [10], for example, an interposed coordinator bridges the standard OTS two-phase commit protocol from a CORBA object or EJB on Unix or Windows to the proprietary Resource Recovery Management Services (RRMS) two-phase commit protocol on the mainframe. Bi-directional transactional interoperability with CICS and IMS is achieved using an interposed coordinator on the mainframe to map the standard OTS two-phase commit commands into and out of their RRMS equivalents. The standard OTS protocol is used over the wire.

The OASIS WS-AtomicTransaction and WS-BusinessActivity specifications, which share some of the same authors as WS-CAF, have recently reached OASIS standards. They are roughly equivalent to WS-ACID and WS-LRA. They also share the common ancestry of the CORBA Activity Service.

4 Conclusions

Specifications such as the Business Process Execution Language (BPEL) and the Web Services Choreography Interface (WSCI) focus on tying multiple Web services together to create multi-step applications, such as filling a purchase order or resolving an insurance claim. Therefore these applications have the requirement to share context across the steps.

The WS-CAF specifications define a standard framework for use by a set of cooperating Web services so that:

- Each Web service knows what application it's included in (or how many and which one it's currently in).
- The Web services in a composite have a way to obtain results of another Web service's operations.
- A standard mechanism is available to share needed system data such as security tokens, file and device handles, or network addresses.
- The application can set rules and policies for recovering from the failure of one or more of the services.

While specifications such as BPEL and WSCI provide the mechanism for extending the WSDL layer to identify a series or sequence of execution for multiple Web services, WS-CAF defines the complementary system layer necessary to ensure that the multiple Web services achieve the desired results of the application, and that the cooperation of multiple Web services from whatever source (local or remote) produces predictable behavior despite system failure and leaves the system in a known state.

As with most aspects of standardization, the value in WS-CAF is derived from the potential for its features and functions to be provided by Web services vendors, therefore helping application developers solve composite application problems more easily. Once adopted and implemented, the functionality contained within WS-CAF will not only be available as part of the platform (and therefore not have to be coded as part of the application) but also it will be available in a standard way across platforms, allowing Web services from multiple environments to interoperate more easily, efficiently, and effectively than if the developers had to code all of the equivalent features and functionality themselves in a non-standard way.

References

[1] RFC 237, 1998 http://www.faqs.org/rfcs/rfc2371.html
[2] OASIS Web Services Composite Application Framework Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf
[3] OASIS Web Services Business Process Execution Language Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
[4] The Web Services Choreography Interface, August 2002, W3C Note, http://www.w3.org/TR/2002/NOTE-wsci-20020808/
[5] W3C Architecture Committee, see http://www.w3.org/2002/ws/arch/
[6] http://www-306.ibm.com/software/solutions/webservices/pdf/SecureReliableTransactedWSAction.pdf
[7] See http://www.ws-i.org/Documents.aspx for background on WS-I and information on WS-I working group charters
[8] http://www.omg.org/technology/documents/formal/add_struct.htm
[9] http://www.omg.org/technology/documents/formal/transaction_service.htm
[10] http://www.iona.com/products/appserv-mainframe.htm
[11] Session Modeling for Web Services, Hal Hildebrand et al, Proceedings of ECOWS (2005)
[12] W3C WS-Addressing Working Group, http://www.w3.org/2002/ws/addr/

Resilient Security for False Event Detection Without Loss of Legitimate Events in Wireless Sensor Networks

Yuichi Sei[1,2] and Shinichi Honiden[1,3]

[1] Graduate School of Information Science and Technology
The University of Tokyo
731 Hongo, Bunkyoku Tokyo, Japan
[2] Research Fellow of the Japan Society for the Promotion of Science (JSPS)
[3] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
{sei, honiden}@nii.ac.jp

Abstract. When large-scale wireless sensor networks are deployed in hostile environments, the adversary may compromise some sensor nodes and use them to generate false sensing reports or to modify the reports sent by other nodes. Such false events can cause the user to make bad decisions. They can also waste a significant amount of network resources. Unfortunately, most current security designs have drawbacks; they either require their own routing protocols to be used, or lose legitimate events stochastically and completely break down when more than a fixed threshold number of nodes are compromised. We propose a new method for detecting false events that does not suffer from these problems. When we set the probability of losing legitimate events to 1%, our proposal method can detect more false events than related method can. We demonstrate this by mathematical analysis and simulation.

Keywords: wireless sensor network, false event, security, en-route filtering, node compromise.

1 Introduction

A core function of wireless sensor networks is to detect and report events. Such networks are suitable for tasks such as military surveillance and forest fire monitoring and deploy a large number of sensor nodes over a vast region. Sensor nodes detect events of interest and deliver reports to the *sink* over multihop wireless paths. However, an adversary may capture and compromise several sensors. He can obtain the secret keys stored in the compromised nodes, and these nodes can then pretend to have detected a nearby event or forward a report supposedly originating from a remote location (Fig. 1).

Several studies [1,2] have proposed mechanisms to enable message authentication in sensor networks. However, these mechanisms can only prevent false

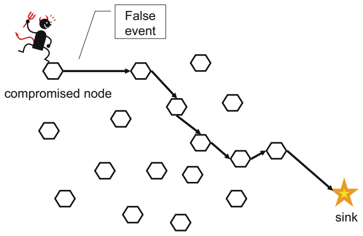


Fig. 1. Compromised node can be used by an attacker to report a false event

reports by outside attackers. They cannot block false reports by compromised nodes.

Work that addresses insider attacks can be classified by mechanism into two types: randomized key distribution mechanisms [3,4,5] and location-based key distribution mechanisms [6,7].

The common goal of these works is to detect false events in networks as quickly as possible. Detection in the early stages is important because multihop transmission wastes the finite resources of sensor nodes.

However, the current mechanisms have drawbacks. Randomized key distribution mechanisms lose legitimate events stochastically and exhibit a threshold behavior. This design is secure against T or less compromised nodes, where T is a fixed threshold, but when more than T keys are leaked, the mechanism cannot detect false events created by using the arbitrary T keys.

Location-based key distribution mechanisms, in contrast, specify the routing protocol of events. A lot of research has been done on routing protocols for wireless sensor networks. The best protocol should be chosen based on various measures such as density of nodes and energy consumption and not simply on the basis of ability to detect false events.

We propose a method for detecting false events that neither loses legitimate events nor specifies routing protocols. We also seek to overcome the threshold limitation of randomized key distribution algorithms. To this end, we introduce a novel notion: *detecting sensor group* (DSG). The nodes within a DSG cooperate and generate keys for each node. Our method also manages the possible combinations of keys attached to a legitimate event to increase the tolerance threshold for number of compromised nodes. Our goal is illustrated in Table 1.

When we set the probability of losing legitimate events to 1%, our method is able to detect more false events than related methods (which do not specify routing protocols) can. We demonstrate this with mathematical analysis and simulations.

The rest of the paper is organized as follows. Section 2 presents a model of false events and sensor networks. Section 3 discusses related methods and their problems. Section 4 presents the design of our algorithm. Section 5 discusses parameter setting and the analyses of our algorithm's effectiveness through mathematical analysis. Section 6 presents the results of simulations of our design. A number of practical issues are discussed in Section 7, and Section 8 concludes the paper.

Table 1. Current methods

Method \ Characteristics	Not losing legitimate events	Not specifying routing protocols	Tolerant of more than T compromised nodes
Randomized key distribution		X	
Location-based key distribution	X		X
Our method	X	X	X

2 System Models

In this section, we define our assumed sensor network model and the model of false event attacks.

2.1 Sensor Network Model

We assume a sensor network composed of a large number of small sensor nodes deployed at high density. By detecting an event using several nodes, such a network can improve accuracy detection and adapt to node failures. Each of the detecting nodes reports the signal it senses, and one of them is chosen as the center-of-stimulus node (CoS). The CoS gathers and summarizes all the received detection results. The message from the CoS to the sink, which is called the *event message*, travels to the sink over a multihop route. The sink is a data collection center with large computation and storage capabilities that protects itself using advanced security solutions. Because they are designed to be inexpensive, we assume that the sensor nodes are not equipped with tamper-resistant hardware. Once deployed, each node can determine its geographic location using a localization scheme [8,9].

2.2 False Event Attack

An attacker may compromise multiple sensor nodes in the network. Once a sensor node is compromised, all secret keys, data, and code stored on it are exposed to the attacker. The attacker can load a compromised node with secret keys obtained from other nodes. The compromised nodes can pretend to have detected an event nearby. Such bogus reports can cause the user to make bad decisions and cause the failure of mission-critical applications. They can also induce congestion, and waste a significant amount of network resources (e.g., energy and bandwidth), along data delivery paths.

The compromised nodes can launch many other attacks. However, these threats are addressed in other related work [10,11] and are beyond the scope of this paper.

3 Related Work

Although many researchers address the method of distributing keys to nodes in safety, these methods [12,13] do not consider situations where sensor nodes are compromised and the information stored in them is leaked.

There are several studies on the similar topic of secured data aggregation [14,15]. These methods presented in these studies can detect false events at the sink but cannot detect them in-network.

Research on false event detection in-network has devised a way to distribute secret keys to sensors [3,4,5,6,7]. The basic idea is allowing every node have symmetric keys. In a randomized key distribution mechanism, every node is preloaded with them. In a location-based mechanism, key are generated after deployment based on each node's location. When an event occurs, multiple surrounding sensors collectively generate a report that carries multiple message authentication codes (MACs) [16]. $MAC(K, M)$ is the MAC of message M using key K. A MAC is generated by a node using one of its symmetric keys and represents the node's agreement on the report. As a report is forwarded toward the sink over multiple hops, each forwarding node probabilistically verifies the correctness of the MACs carried in the report. Reports with inadequate numbers of MACs are not delivered.

These methods have the drawbacks of losing legitimate events stochastically or needing to use their own routing protocols.

There are several studies that have expanded on these researches. For example, some researchers [17,18] have applied a re-keying technique to the existing detection mechanism [3]. F. Li [19] addresses a fabricated report with a false MAC attack. The relationship between the false event detection methods and the re-keying techniques is shown in Figure 2. Re-keying and false MAC techniques use false event detection mechanism without modification and add additional functions. Re-keying and false MAC techniques add functions to the unmodified false event detection mechanism. In the figure, the reference relationships are represented as arrows. We propose a detection mechanism that we can combine with the re-keying or the unmodified false MAC techniques.

3.1 Problems of a Randomized Key Distribution Mechanism

F. Ye et al. [3] and H. Chan et al. [4] distribute keys to the sensor nodes randomly. Sensor nodes need to collect more than T discrete MACs to create a legitimate event. However, they may fail to collect them even if more than T legitimate nodes detect an event. In Fig. 3, which shows an example where T=3, sensors are represented as hexagons. The center-of-stimulus nodes can collect only two types of MACs although the system needs more than three MACs. In this case, the event message to the sink will be dropped by an intermediate node.

Moreover, a randomized key distribution mechanism exhibits a threshold behavior. False events cannot be detected when the attacker has more than T keys. A false event could appear either where nodes are compromised or at arbitrary locations.

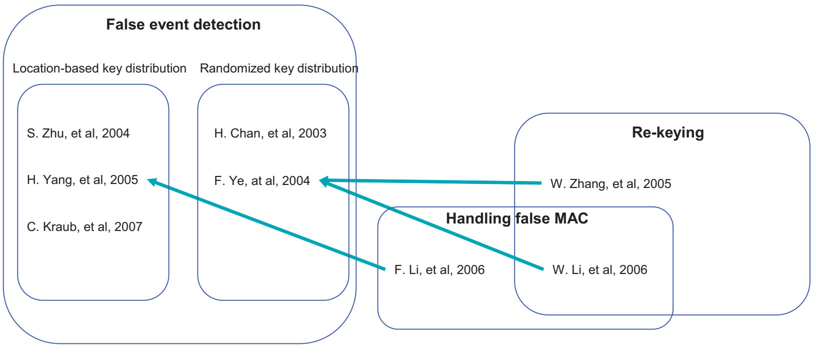


Fig. 2. Work related to false event detection

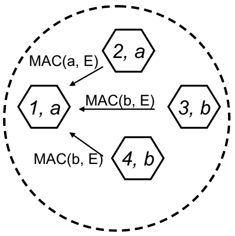


Fig. 3. Losing a legitimate event (Numbers represent node IDs, and letters represent ID of key held by node)

3.2 Problems of a Location-Based Key Distribution Mechanism

H. Yang et al. [5], S. Zhu et al. [6], and C. Kraub et al. [7] distribute keys to each sensor node based on its location. The methods described in these papers restrict the routing path of an event message to boost the probability of false event detection.

However, there has been a lot of research on the routing protocols of wireless sensor networks, which enables designers to choose the best protocol based on various measures (density of nodes, energy consumption, and so on). Choosing a routing protocol simply to detect false events ignores these other benefits.

4 Design

4.1 Definition of Novel Concepts and Notations

Four concepts form the basis of our method: detecting sensor group (DSG), minimum sensor group (MG), legitimate combination of keys (LCK), and event circle region (ECR). Sensor nodes are represented as n_i, where of n_i's node ID is i.

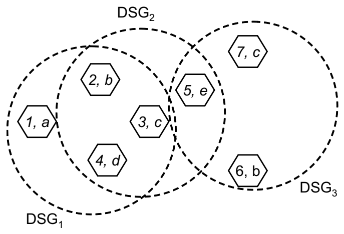


Fig. 4. Detecting sensor group (DSG) where T=3

Detecting Sensor Group (DSG). We assume ideal sensing within sensing range Rd for each sensor. In ideal sensing, targets are detectable if the sensors are within a certain range of the target and no signal is received by a sensor outside this range [20].

A DSG is:

1. A combination of more than T sensors included in a circle with radius r_d
2. Not a subset of any other DSG

Figure 4 shows an example where $T = 3$ and each sensor is assigned an ID. The circles in the figure have radius r_d. The set of sensors in a given circle is a DSG. Let the node whose ID is i be n_i. DSG_1 is $\{n_1, n_2, n_3, n_4\}$, DSG_2 is $\{n_2, n_3, n_4, n_5\}$, and DSG_3 is $\{n_5, n_6, n_7\}$.

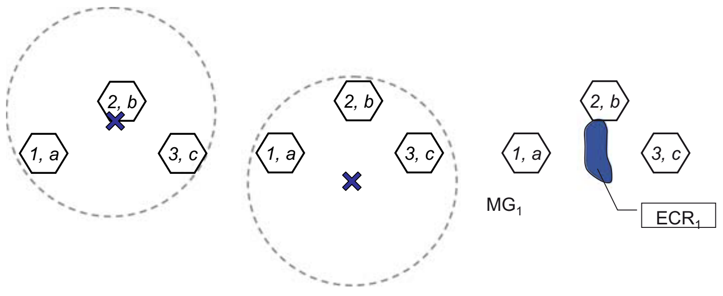
If n_1 does not exist, the three nodes included in DSG_1 are all included in DSG_2. In this case, the set of nodes $\{n_2, n_3, n_4\}$ is not a DSG because it does not meet the second definition of DSGs.

The number of nodes in a DSG must be larger than T to create a legitimate event. Sensor nodes in the same DSG must have different keys so as not to lose a legitimate event.

Minimum Sensor Group (MG) and Legitimate Combination of Keys (LCK). A minimum sensor group (MG) is defined as the combination of T nodes in the same DSG. For example, in Fig. 4, there are eight MGs: $MG_1 = \{n_1, n_2, n_3\}, MG_2 = \{n_2, n_3, n_4\}, MG_3 = \{n_3, n_4, n_1\}, MG_4 = \{n_4, n_1, n_2\}$, $MG_5 = \{n_3, n_4, n_5\}, MG_6 = \{n_4, n_5, n_2\}, MG_7 = \{n_5, n_2, n_3\}, MG_8 = \{n_5, n_6, n_7\}$.

Each legitimate combination of keys (LCK) is associated with an MG. Let the ID of n_i's key be $k(n_i)$. For example, LCK_1, which is associated with MG_1, is $\{k(n_1), k(n_2), k(n_3)\} = \{a, b, c\}$. The key IDs of an LCKs are potential legitimate combinations of key IDs of legitimate events. For example, it is possible for an event message to have T MACs created by key IDs a, b, and c, but it is not possible for it to have MACs created by key IDs a, c, and e.

Event Circle Region (ECR). Each MG has its event circle region (ECR). Figures 5(a) and 5(b) show example of ECRs. The x represents the location of



(a) Possible detecting location 1 (b) Possible detecting location 2 (c) Entire possible detecting location (ECR)

Fig. 5. Event circle region associated with MG_1

an event, and the circle's radius is r_d. Only the nodes in the circle can detect the event. $MG_1 = \{n_1, n_2, n_3\}$ can detect an event only in the region named ECR_1 in Fig. 5(c).

4.2 Overall Operations

In our method, as in other current methods, we let each node have a key. However, we also let each node in the same DSG have a different key so as not to lose legitimate events. We assume that sensor nodes' locations are not known a priori. Since they may have been dropped by vehicles or aircraft, their locations must be determined before reports can be processed. Positive determination of a node's location is required in all applications used in a hostile environment. A number of proposals [8,9] have started to address this problem. After their locations have been determined, sensor nodes need to recognize what DSGs they are in. We introduce an efficient way for the system to determine this.

However, we also need a mechanism by which we can detect a false event even if more than T nodes are compromised. To do this, we use LCKs, i.e., potential legitimate combinations of key IDs of legitimate events.

For example, for the configuration shown in Fig. 4, let the attacker compromise three keys k_a, k_b, and k_c and create a false event using the keys. Here, k_i is the key whose ID is i. If we use a randomized key distribution mechanism, the false event may appear both where nodes are compromised, and at arbitrary locations. However, if the sink knows all ECRs and corresponding LCKs, it can detect a false event that occurred in an ECR not associated with the reporting LCK.

Deriving Keys in an Efficient Fashion. We preload each node with the number of entire keys M, a master secret key $\hat{K}$, and a secure one-way hash function $H(\cdot)$ [21] before it is deployed. M is determined based on its power to detect false events and its tolerance of compromising nodes. We discuss this further in Section 5.

Once deployed, a node n_i first determines its geographic location L_i through a localization scheme [8,9]. Then n_i broadcasts information about i, and L_i within

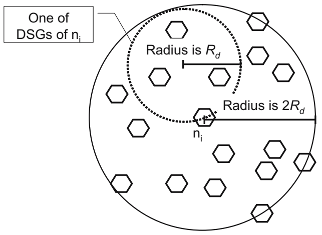


Fig. 6. DSGs that may include n_i must be within a circle of radius $2 \cdot R_d$

a $2 \cdot R_d$ radius using the method described in [11]. The DSGs in which n_i can possibly be included are within the circle with a $2 \cdot R_d$ whose center is L_i (see Fig. 6). Assume the node n_j obtained information from ϵ nodes. DSGs that include n_j potential are limited to the combinations of those ϵ nodes. Therefore, each node can determine the DSGs it belongs to using the ϵ information.

In each DSG, the node with the largest node ID will be the leader of the DSG. Each leader sends the information to the sink. The sink can determine locations of all nodes and DSGs. Then the sink can calculate all MGs, LCKs, and ECRs. It stores all the information.

Each node n_i follows the procedure below:

– If n_i is not a leader of a DSG, it deletes the master secret $\hat{K}$.
– It finds the leader with the largest node ID among the leaders of all of its DSGs (The leader maybe n_i itself).
– n_i requests a key from the leader with the largest node ID and receives a key ID and a corresponding key from the leader.
– It then registers the key ID with the leaders of its other DSGs.

Each DSG leader node also executes the above procedures. The additional procedures for each leader node n_l are as follows:

– It receives a key ID request from each node in the same DSG.
– If n_l receives the requests or key registrations from all nodes in one DSG (n_l may belong to several DSGs), it determines each key ID. Let the number of nodes in the DSG be s. Node n_l calculates $k_{id} = \lfloor M \times \text{random}() \rfloor$ repeatedly until it gets s different results. The function random() returns a value with a positive sign greater than or equal to 0.0 and less than 1.0. Then it calculates $k = H_{\hat{K}}(k_{id})$ from each k_{id}.
– n_l then sends each key ID k_{id} and the corresponding key k to the nodes that requested keys in the DSG.
– When n_l finishes giving key IDs to the nodes that requested keys in DSGs where it is a leader, it deletes the master secret $\hat{K}$ immediately.

As a result, all sensor nodes in the same DSG are assigned different key IDs.

Event Generation. When an event occurs, a center-of-stimulus node is elected using the method proposed in [11]. Then, all surrounding nodes that detect the signal will prepare an event report in the form of $\{L_E, t, E\}$. L_E is the location of the event, t is the time of detection, and E is the type of event. Although the report may also contain other information, we only list these three, as described in [3] to simplify presentation. Then a node n_i generates:

$$MAC_i = \overline{MAC}(k(n_i), L_E || E) \tag{1}$$

where $||$ denotes stream concatenation. $k(n_i)$ represents the key n_i has.

The node n_i then sends $\{k_{id}(n_i), MAC_i\}$, the key index of $k(n_i)$ and the MAC, to the center-of-stimulus node. The center-of-stimulus node collects all the $\{k_{id}(n_i), MAC_i\}$ from detecting nodes. The final report sent out by the center-of-stimulus node to the sink is

$$\{L_E, t, E, k_{id}(n_{i_1}), MAC_{i_1}, ..., k_{id}(n_{i_T}), MAC_{i_T}\}.$$

En-Route Filtering. This phase is similar to current methods, such as that presented in [3]. We give each sensor node a simple additional task. The task of an intermediate node is as follows.

When a node receives an event message, it first examines whether there are T key IDs and T MACs in the message. Messages with less than T key IDs or less than T MACs are dropped. Then the node executes an additional task: it examines whether the IDs of T keys are between 0 and $M - 1$. If any of them are not in the range, the event message is dropped. This examination must be done to prevent an attacker from attaching non-existing values to the message as IDs. In current methods sensor nodes do not do this examination.

Then, if the node has any of the T keys indicated by the key IDs, it reproduces the MAC using its own key and compares the result with the corresponding MAC attached in the message. The message is dropped if the attached one differs from the reproduced one. If they match exactly, or this node does not possess any of the T keys, the node passes the message to the next hop.

Sink Verification. When the sink receives an event message, it can check the correctness of every MAC_{i_j} because it has all the keys. Any forged MAC that chances to get behind the en-route filtering will be caught. If the message is considered a false event, it is discarded.

Then, the sink examines whether or not the set of key IDs in the message is one of the legitimate combinations of keys (LCKs). If the examination failed, it can be concluded that the message was created by an attacker who had gathered T keys from discretely located nodes.

Then, the sink examines whether or not location information L_t in the message is included in the corresponding event circle region (ECR). If the examination failed, it can be concluded that the location information was forged by an attacker. Therefore, the message is discarded.

Handling of Node Movement. If a sensor node moves from its location, the sink should know this. The process for the sink to discover this is illustrated in Fig. 7 (where the leader node of each DSG is colored in gray) and is as follows.

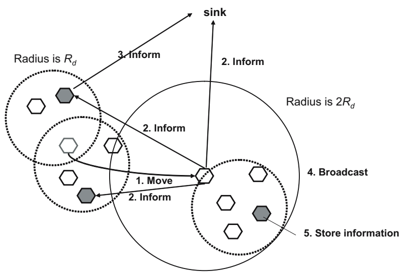


Fig. 7. Process for handling node movement

- Moved node n_i informs the leader nodes of its previous DSGs and the sink of it's new location.
- n_i's previous DSG leader nodes check whether their DSGs have still enough nodes to create legitimate events. If not, they inform the sink of this problem. The sensor network manager may want to add a sensor node to that location.
- Moved node n_i broadcasts its ID and new location within a $2 \cdot R_d$ radius circle.
- n_i's new DSG leader nodes store the ID and the location.

n_i does not need a new key because the DSGs that contain n_i already have enough keys to create legitimate events even if n_i's key overlaps with other nodes' keys.

5 Analysis

We start the analysis of the performance of our design by examining the filtering power of our design against compromised nodes. Then, we analyze its resiliency as more and more nodes are compromised.

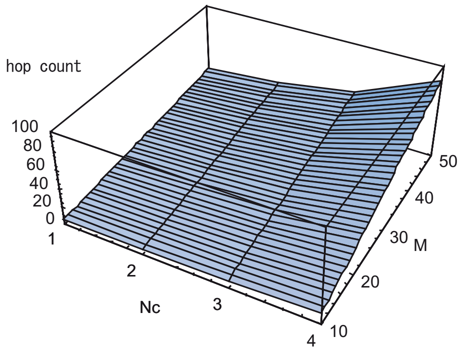
5.1 Filtering Effectiveness

We first consider how many keys are compromised if an attacker compromises N_c nodes (even if N_c sensor nodes are compromised, the number of leaked keys is less than N_c, because some nodes share keys). Let the number of leaked keys be N_k.

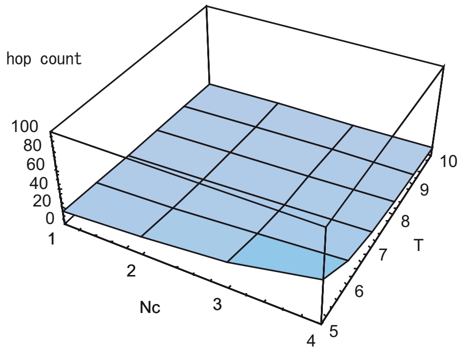
The relationship between N_c and N_k is shown as:

$$\begin{aligned} N_c &= M\left(\frac{1}{M} + \frac{1}{N-1} + \cdots + \frac{1}{M-N_k}\right) \\ &\approx M \ln\left(\frac{M}{M-N_k}\right), \end{aligned} \quad (2)$$

where M represents the number of keys.



(a) $T = 5$



(b) $M = 30$

Fig. 8. Number of hops traveled

We can get from Equation 2:

$$N_k = M \cdot e^{-N_c/M} \cdot \left(e^{N_c/M} - 1\right) \tag{3}$$

The probability that a node can detect a false event when the attacker obtained $N_k (N_k < T)$ keys, denoted by p_1 is

$$p_1 = \frac{T - N_k}{M} \tag{4}$$

The average hop count until the false event is detected, denoted by p_h is

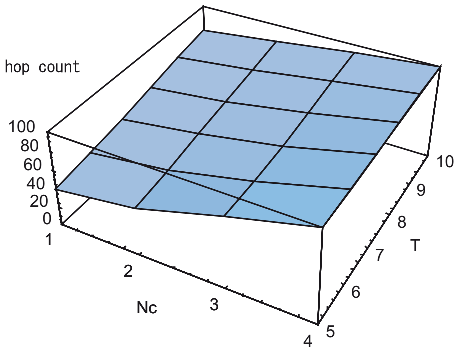
$$\begin{aligned} p_h &= \sum_{i=1}^{H} i \cdot (1 - p_1)^{i-1} \cdot p_1 \\ &= \frac{1 - (1 - p_1)^H}{p_1}, \end{aligned} \tag{5}$$

where H represents the max hop count from the center-of-stimulus node to the sink.

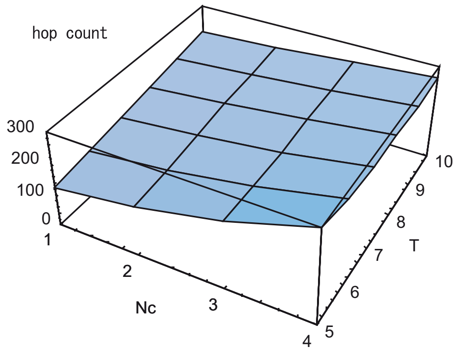
Figure 8 shows the results of the analysis. Consider an example of $H = 500$. N_c is the number of compromised nodes, T is the number of MACs carried in an event message, and M is the number of keys.

Figure 8(a) shows that the filtering power grows as M decreases. We know that most false events are dropped within ten hops if the attacker has one key when M is 30 and T is 5 in the example in Fig. 8(b). In the worst case, where only one MAC is incorrect , the false events travel an average of 24 hops. Note that the filtering power does not depend on the number of sensor nodes.

Next we analyze the filtering power of SEF [3]. SEF is one of the methods that use a randomized key distribution mechanism. Many recent methods use the SEF algorithm without modification (and add extra functions to it). We



(a) Probability of losing legitimate events is 1%



(b) Probability of losing legitimate events is 0.1%

Fig. 9. Number of hops traveled in SEF

think we should compare our work with SEF. Note that the methods that use the SEF algorithm can also use our algorithm without modification instead of SEF.

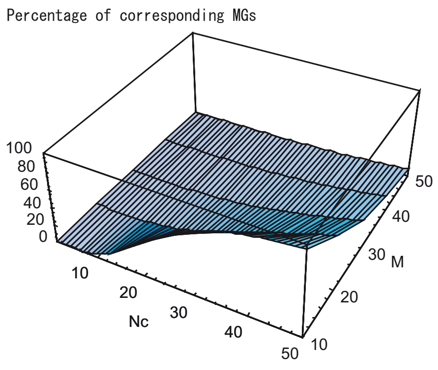
We used parameters of T and H and set the number of nodes in each DSG to $T + 1$. When using SEF, we need to determine the number of *key partitions*. If this parameter is small, filtering power increases but the probability of losing legitimate events also increases. We set it based on the target probability of losing legitimate events. In regard to other parameters, we followed [3]. The results are shown in Fig. 9. In Fig. 9(a) we set the probability of losing legitimate events at 1%, and in Fig. 9(b) we set it at 0.1%. We know that SEF is of limited use if we do not want to lose legitimate events. Even when the probability is 1%, our method is more effective than SEF as shown in Fig. 8(b) and Fig. 9(a).

5.2 Resiliency to Compromising Many Nodes

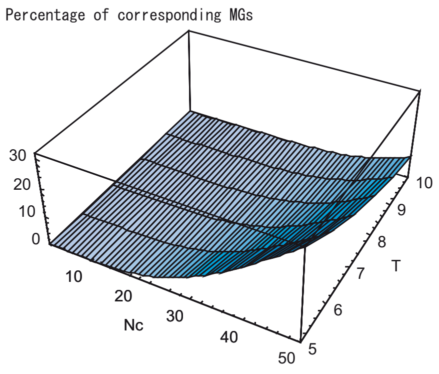
Now we consider the resiliency of our method to attacks where an increasing number of nodes are compromised. We consider a case where the attacker compromises N_c nodes and fabricates event messages on bogus events happening in an arbitrary location. To fabricate events without being detected, the attacker must collect T keys of one LCK.

Let the ratio of the geographical region where we cannot detect false events when an attacker compromised N_c sensor nodes be $\Re$. If the attacker obtained N_k keys, he can create ${}_{N_k}C_T$ combinations of legitimate T keys. If the number of all keys is M, the number of the possible combinations of legitimate keys is ${}_MC_T$. There are many minimum sensor groups in the network and each group is associated with one LCK. Therefore, $\Re$ is shown as follows:

$$\Re = \frac{{}_{N_k}C_T}{{}_MC_T} \tag{6}$$



(a) $T = 5$



(b) $M = 30$

Fig. 10. Relationship between number of compromised nodes and corresponding minimum sensor groups

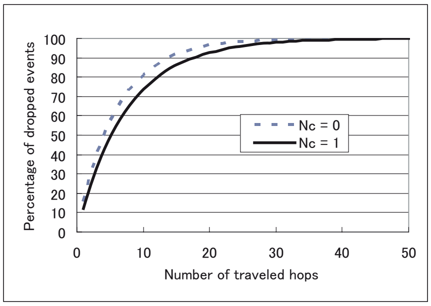
Figure 10 shows the analysis results for $\Re$. As can be seen in Fig. 10(a), $\Re$ decreases as M increases. When M is 30, $\Re$ is only 30%, even if the number of compromised nodes is 50. Note that in SEF, $\Re$ is almost 100% when T is 5 and there are only five compromised nodes.

6 Simulation Evaluation

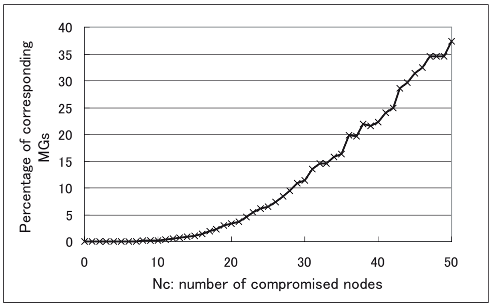
We use simulations to further verify our analysis. We developed our own simulation platform, mainly because other simulators scale poorly to large numbers of nodes. Our simulator implemented basic geographic forwarding [22]. We used a field size of $1000 \times 100m$ in which 10,000 nodes were uniformly distributed. There are six nodes in an event circle region, and T is 5. There was one stationary sink and one stationary source at opposite ends of the field, with about 500 hops between them.

Figure 11(a) shows the percentage of dropped false reports as a function of number of traveled hops, for 0 and 1 compromised nodes, respectively. The source generates 1,000 false events in each run. When the attacker mimics wireless transmission to inject traffic, about 80% of false events are dropped within ten hops. With one compromised node, about 70% are dropped within ten hops.

Figure 11(b) shows the relationship between the number of compromised nodes and the corresponding minimum sensor groups, i.e., the ratio of the geographical region in which we cannot detect false events when an attacker has compromised N_c sensor nodes. In the simulation, T was set to 5 and M was set to 30. The attacker randomly picked up N_c nodes. We checked the number of minimum sensor groups that the attacker can mimic from the obtaining keys. Then we divided the resulting value by the number of all minimum sensor groups. We ran 1,000 simulations. We know that the percentage of corresponding minimum sensor groups is only 37% even if there are 50 compromised nodes.



(a) Number of hops



(b) Relationship between number of compromised nodes and corresponding minimum sensor groups

Fig. 11. Simulation results

7 Discussion

In addition to sensing data, each event message carries T key IDs and MACs. To reduce the MAC size, we can use data structures named Bloom Filter [23] or Ringed Filter [24] in the same way [3] does. These are space-efficient probabilistic data structures used to test whether or not an element is a member of a set.

Currently our work does provide no key update or revocation mechanism. Recent work [17,18] has started to address sensor re-keying, so we can update or revoke the keys either periodically or when there are security breaches.

We assume ideal sensing within a sensing range for each sensor. In practice, sensing is noisy. A sensor can detect an event outside its nominal range or may fail to detect an event inside its range. Our approach to such non-idealities uses a sensing model in which the event is always detected within an inner disk of radius r_d, called the detection region and is detected with some nonzero probability in an annulus between the inner disk and an outer disk of radius r'_d called the uncertain region. Events outside the outer disk are never detected. This uncertainty has an effect on creation of detecting sensor groups (DSGs). At this time, each node n_i needs to broadcast for creating DSGs within a $2 \cdot R'_d$ radius rather than $2 \cdot R_d$ radius. Note that this does not influence the en-routing filtering power of our algorithm.

Currently, our design does not address identifying compromised sensor nodes, which may be necessary for the continuous operation of a network. To identify the compromised nodes, each node can use the watchdog mechanism [25] to monitor its neighbors and identify the compromised nodes when observing misbehavior. The collaborative intruder identification scheme [26] can also be used to improve accuracy.

8 Conclusion

Compromised nodes present severe security threats in sensor networks. Current solutions either require their own routing protocols to be used or lose legitimate events stochastically and completely break down when more than a fixed threshold number of nodes is compromised. In this paper, we introduced new notations: *detecting sensor groups* and *possible legitimate combinations of keys*. We proposed an efficient way of creating node keys in the same detecting sensor group to avoid losing legitimate events. We also proposed a way of detecting false events by managing possible legitimate combinations of keys, even if an attacker has compromised many nodes. When we set the probability of losing legitimate events to 1%, our proposal method can detect more false events than related methods can. We could also limit the region where the attacker can create undetectable false events by compromising many sensor nodes. In future work, we plan to implement our algorithm in sensor nodes.

Acknowledgments

We thank our colleagues in X-Aware Computing (XAC) at the National Institute of Informatics for their constructive criticism.

References

1. Perrig, A., Szewczyk, R., Wen, V., Culler, D.E., Tygar, J.D.: SPINS: security protocols for sensor networks. In: Mobile Computing and Networking (MobiCom), pp. 189–199 (2001)
2. Eschenauer, L., Gligor, V.D.: A key-management scheme for distributed sensor networks. In: CCS 2002. Proceedings of the 9th ACM conference on Computer and communications security, pp. 41–47. ACM Press, New York, NY, USA (2002)
3. Ye, F., Luo, H., Lu, S., Zhang, L.: Statistical en-route filtering of injected false data in sensor networks. IEEE Journal on Selected Areas in Communications, Special Issue on Self-organizing Distributed Collaborative Sensor Networks 23(4), 839–850 (2005)
4. Chan, H., Perrig, A., Song, D.: Random key predistribution schemes for sensor networks. In: Proceedings of the 2003 IEEE Symposium on Security and Privacy, pp. 197–213. IEEE Computer Society Press, Washington, DC, USA (2003)
5. Yang, H., Ye, F., Yuan, Y., Lu, S., Arbaugh, W.: Toward resilient security in wireless sensor networks. In: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing, pp. 34–45. ACM Press, New York, NY, USA (2005)
6. Zhu, S., Setia, S., Jajodia, S., Ning, P.: An interleaved hop-by-hop authentication scheme for filtering false data injection in sensor networks. In: Proceedings of IEEE Symposium on Security and Privacy, pp. 259–271. IEEE Computer Society Press, Los Alamitos (2004)
7. Kraub, C., Schneider, M., Bayarou, K., Eckert, C.: Stef: A secure ticket-based en-route filtering scheme for wireless sensor networks. In: ARES 2007. The Second International Conference on Availability, Reliability and Security, pp. 310–317. IEEE Computer Society Press, Los Alamitos, CA, USA (2007)

8. Bruck, J., Gao, J., Jiang, A.A.: Localization and routing in sensor networks by local angle information. In: MobiHoc 2005. Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing, pp. 181–192. ACM Press, New York, NY, USA (2005)
9. Stoleru, R., Vicaire, P., He, T., Stankovic, J.A.: Stardust: a flexible architecture for passive localization in wireless sensor networks. In: SenSys 2006. Proceedings of the 4th international conference on Embedded networked sensor systems, pp. 57–70. ACM Press, New York, NY, USA (2006)
10. Wood, A.D., Stankovic, J.A.: Denial of service in sensor networks. Computer 35(10), 54–62 (2002)
11. Ye, F., Zhong, G., Lu, S., Zhang, L.: Gradient broadcast: a robust data delivery protocol for large scale sensor networks. Wirel. Netw. 11(3), 285–298 (2005)
12. Liu, D., Ning, P.: Establishing pairwise keys in distributed sensor networks. In: CCS 2003. Proceedings of the 10th ACM conference on Computer and communications security, pp. 52–61. ACM Press, New York, NY, USA (2003)
13. Du, W., Deng, J., Han, Y.S., Chen, S., Varshney, P.K.: A key management scheme for wireless sensor networks using deployment knowledge. In: INFOCOM (2004)
14. Yang, Y., Wang, X., Zhu, S., Cao, G.: Sdap: a secure hop-by-hop data aggregation protocol for sensor networks. In: MobiHoc 2006. Proceedings of the seventh ACM international symposium on Mobile ad hoc networking and computing, pp. 356–367. ACM Press, New York, NY, USA (2006)
15. Chan, H., Perrig, A., Song, D.: Secure hierarchical in-network aggregation in sensor networks. In: CCS 2006. Proceedings of the 13th ACM conference on Computer and communications security, pp. 278–287. ACM Press, New York, NY, USA (2006)
16. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-hashing for message authentication IETF - Network Working Group, RFC2104 (February 1997)
17. Zhang, W., Cao, G.: Group rekeying for filtering false data in sensor networks: a predistribution and local collaboration-based approach. In: INFOCOM, pp. 503–514 (2005)
18. Li, W., Zhang, Y., Yang, J.: Dynamic authentication-key re-assignment for reliable report delivery. In: IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS), pp. 467–476. IEEE Computer Society Press, Los Alamitos (2006)
19. Li, F., Wu, J.: A probabilistic voting-based filtering scheme in wireless sensor networks. In: Proceeding of the 2006 international conference on Communications and mobile computing, pp. 27–32. ACM Press, New York, NY, USA (2006)
20. Shrivastava, N., Madhow, R.M.U., Suri, S.: Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In: SenSys 2006. Proceedings of the 4th international conference on Embedded networked sensor systems, pp. 251–264. ACM Press, New York, NY, USA (2006)
21. Tsudik, G.: Message authentication with one-way hash functions. SIGCOMM Comput. Commun. Rev. 22(5), 29–38 (1992)
22. Karp, B., Kung, H.T.: GPSR: Greedy perimeter stateless routing for wireless networks. In: Mobile Computing and Networking (MobiCom), pp. 243–254 (2000)
23. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13(7), 422–426 (1970)

24. Sei, Y., Matsuzaki, K., Honiden, S.: Ringed filters for peer-to-peer keyword searching. In: ICCCN. IEEE 16th International Conference on Computer Communications and Networks, IEEE Computer Society Press, Los Alamitos (to appear, 2007)
25. Marti, S., Giuli, T.J., Lai, K., Baker, M.: Mitigating routing misbehavior in mobile ad hoc networks. In: MobiCom 2000. Proceedings of the 6th annual international conference on Mobile computing and networking, pp. 255–265. ACM Press, New York, NY, USA (2000)
26. Wang, G., Zhang, W., Cao, G., Porta, T.: On supporting distributed collaboration in sensor networks. In: IEEE Military Communications Conference (MILCOM) (2003)

Formal Verification of a Group Membership Protocol Using Model Checking

Valério Rosset, Pedro F. Souto, and Francisco Vasques

Faculdade de Engenharia, Universidade do Porto
Rua Dr. Roberto Frias s/n
4200-465 Porto, Portugal
vrosset@fe.up.pt, pfs@fe.up.pt, vasques@fe.up.pt

Abstract. The development of safety-critical embedded applications in domains such as automotive or avionics is an exceedingly challenging intellectual task. This task can, however, be significantly simplified through the use of middleware that offers specialized fault-tolerant services. This middleware must provide a high assurance level that it operates correctly. In this paper, we present a formal verification of a protocol for one such service, a Group Membership Service, using model checking. Through this verification we discovered that although the protocol specification is correct, a previously proposed implementation is not.

1 Introduction

Safety critical applications in the avionics and in the automotive domains, have extremely demanding reliability requirements that are increasingly being addressed through the adoption of distributed and fault-tolerant architectures. These architectures are usually built on top of specialized middleware that is often integrated with the communications services themselves. For this reason, this middleware is frequently referred to as a *bus* [1]. This is somewhat misleading as it offers rather complex and sophisticated services such as clock synchronization, reliable broadcast and group membership.

Recently, a new *bus*, FlexRay [2], has been proposed for the automotive domain. FlexRay was specified by a consortium of automobile and automotive electronics manufacturers and is likely to become the *de facto* standard of next generation automotive-buses. FlexRay is a minimalist bus in that it provides only communication services and clock synchronization. In a previous paper [3], we presented a new group membership protocol, which we will refer to as *the GMP*, that takes advantage of the dual scheduling ability of the class of TDMA protocols used by FlexRay and argued informally its correctness.

However, fault-tolerant distributed protocols are very subtle and informal arguments are prone to error making them clearly insufficient for safety-critical

[1] Sponsored by scholarship from FCT foundation under reference number SFRH/BD 19302/2004

applications, which require a high level of assurance that they operate correctly. This holds especially for middleware that is supposed to be used in the development of safety critical applications. Ideally, mathematical proofs, either manual or automatic, of its correctness should be provided. An alternative formal method is model checking.

Model checking is a technique for verifying properties of a system through exhaustive and automatic exploration of all the system states. One problem with model checking is the state space explosion, i.e. the exponential growth of the number of system states when the number of components or the number of variables and their possible values increases. A well known technique to address this problem is symmetry reduction [4], which tries to explore the structural symmetry of the model. This technique is particularly effective in models composed by identical components, such as in distributed protocols.

In this paper we focus on the use of symmetry reduction in verifying the GMP. The GMP is particularly challenging in this respect, because its behavior is somewhat "irregular". This is compounded by our desire in keeping the model close to the GMP, in order to ensure a high confidence level on the verification results. Therefore, the model we have developed includes an implementation of the GMP that could be used almost verbatim in an executable implementation of the protocol. This allowed us to detect an error in the outline of an implementation of the protocol that we previously proposed in [3].

The remainder of this paper is organized as follows. In the next section we provide some background information including an informal description of the GMP protocol and a very quick review of UPPAAL, the model checker we use. In Section 3, we describe an UPPAAL model of the GMP. The techniques used to reduce the state space size are described in Section 4. Section 5 presents the correctness properties, and in Section 6 we present and discuss the verification results. Finally, we conclude in Section 7.

2 Background

2.1 Group Membership Protocol

We consider a system composed of a set of nodes, N, that are connected via a broadcast network, in which a node receives every message it broadcasts.

We assume that the broadcast network uses a dual scheduled TDMA protocol such as FlexRay. A dual scheduled TDMA protocol is a variant of the classic TDMA protocol in which the communications cycle is split in two rounds: one whose slots are statically scheduled like in conventional TDMA, and another in which slots are allocated dynamically to nodes. Furthermore, we assume that each node in N has one slot assigned to it in the statically scheduled round, and may be assigned one slot in the dynamically scheduled round, if it so wishes.

Nodes can fail by experiencing one of three types of faults: a *crash fault*, i.e. a node enters a halting state and takes no further action; a *receive fault*, i.e. a fault on reception, that prevents a node from receiving a message broadcasted by another node in that step; a *send fault*, i.e. a fault on broadcasting, that prevents a node from broadcasting a message to the network. Note that receive and send faults do not need to be persistent, e.g. a node may have a receive fault in a round, but be able to receive a message in a subsequent round. We say that a node is non-faulty if it has not experienced any fault since the beginning of the execution, or since it resumed execution, after a fault.

Finally, it is assumed that the communications network is reliable, i.e. it neither looses nor creates/modifies messages.

GMP Overview. Group membership comprises essentially two sub-problems [5]: 1) failure detection; and 2) set agreement. The GMP was designed to keep the solutions to these sub-problems mostly decoupled, and comprises two phases: 1) a failure-detection phase (FD-phase), in which a node determines the operational state of other nodes in the system, and 2) a set agreement phase (SA-phase), in which the non-faulty nodes reach agreement on the operational state of all nodes in the system. (In [3], the SA-phase was called Group Membership.)

In the GMP, each node keeps, among other state, two sets: the M-SET and the M-set. The former is the set of group members and is updated at the end of every SA-phase. The latter is the candidate group membership that is determined by the node during the execution of both GMP phases. In the FD-phase, every group member is required to broadcast a *heartbeat* message, and receives the *heartbeat* messages broadcasted by other nodes. If a group member does not receive the *heartbeat* message from another group member it removes the latter from its M-set. In the SA-phase, every group member is required to broadcast a *vote* message containing the M-set it computed. Then, each group member applies a majority vote on the set of M-sets received during the SA-phase to determine the M-SET.

In order to ensure that the operational state of the nodes in the system is tracked in a timely fashion, the FD-phase is executed in every TDMA cycle. On the other hand, the SA-phase is executed only when events that may lead to a change in the group membership are observed by group-members. This allows the GMP to take advantage of dual-scheduled TDMA protocol: whereas the FD-phase uses the statically scheduled round of the TDMA cycle, the SA-phase uses the dynamically scheduled round of the TDMA cycle, only when events that may lead to group membership change are observed by group members. In a quiescent state, the network bandwidth required for agreement can be used by other aperiodic traffic.

Figure 1 illustrates one possible execution of the GMP for a configuration with 3 nodes, which are assumed to be members of the group at the beginning of the first TDMA cycle, *c*. In that cycle, the 3 nodes execute only the FD-phase, i.e. only send their heartbeat messages. As no event that might lead to a change in the membership is observed by any of the 3 nodes, there is no execution of the SA-phase (equivalently, we say that the SA-phase has no messages). However, in the following TDMA cycle, node 3 has a receive fault on the heartbeat message sent by node 2. As a result, it sends a vote message in the following round (round n+3). The other nodes do not observe any event that might lead to a change in the membership and therefore do not send any vote.

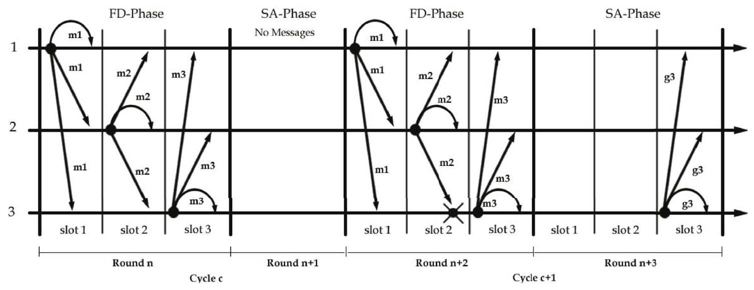


Fig. 1. Possible execution of the GMP, illustrating the execution of the SA-phase only when membership-related events are observed

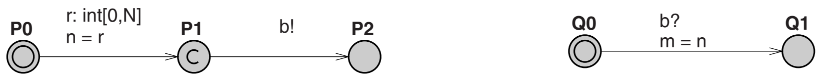


Fig. 2. Simple UPPAAL model composed of two timed-automata

Although the main ideas behind the GMP are simple, the possibility of faults makes the protocol details, that we have omitted, rather subtle. In Annex A, we provide the full protocol for reasons of completeness. A detailed explanation of the protocol, including informal arguments of its correctness, can be found in [3].

2.2 UPPAAL

The UPPAAL model checker [6] is a toolbox for the verification of real-time systems modeled as non-deterministic timed automata. A timed automata [7] is a finite-state machine containing a set of clocks that advance synchronously. UPPAAL supports a number of extensions to timed automata such as integer variables, structured data-types and channel synchronization, that make it suitable to model more than just the temporal behavior of a system.

UPPAAL models comprise a set of timed-automata that execute concurrently and that may synchronize with each other through broadcast or binary channels. Figure 2 shows a simple model with two automata, **P** and **Q**, of three and two locations respectively, i.e. **P0** to **P2** and **Q0** and **Q1**. **P0** and **Q0** are the initial locations of the respective automata and are represented as a double circle. Location **P1**, represented as a circle with a C inside, is a committed location, which means that time is not allowed to advance while such a location is active. The model includes channel `b` and integer variables `m`, `n` and `r`.

Initially, the two automata are in their respective initial location, i.e. **P0** and **Q0**, and all integer variables values are zero. In this state, automaton **Q** cannot take the transition from **Q0** to **Q1** because it is blocked waiting on channel `b`. Therefore, progress is possible only by automaton **P** taking the transition from location **P0** to location **P1**. The edge from location **P0** to location **P1** has a *select*, `r:int[0,N]`, and an *assignment*, `n=r`, labels. The select label binds identifier `r` to a random value in the range `[0,N]`. This value is then assigned to variable `n` in the assignment label. Therefore, when **P** takes transition **P0** to **P1**, variable `n` is assigned a random value in the range `[0,N]`. Because location **P1** is a *committed* location, automaton **P** takes transition **P1** to **P2** immediately after. Simultaneously, automaton **Q** takes transition **Q0** to **Q1**, because both transitions have matching *synchronization* labels on channel `b`. Note that UPPAAL also supports the synchronization of multiple automata on a single *broadcast channel*, i.e. if multiple automata are waiting on a broadcast channel, they will all be unblocked if another automata signals that channel.

A more detailed, and formal, description of UPPAAL can be found, for example, in [8].

3 Verification Model

The UPPAAL model of the GMP comprises two types of automata, or templates: `Node` and `Scheduler`. The `Scheduler` automaton controls the evolution of the protocol by initiating each of the GMP phases. The `Node` automaton models the behavior of one node and is the core of the model. A GMP UPPAAL model comprises one `Scheduler` automaton and N `Node` automata, where N is the number of nodes in the system.

3.1 Basic Model

In order to simplify the presentation of the model we first present a model that does not consider the occurrence of faults.

Global Variables and Synchronization Channels. Variables in UPPAAL may be either local or global. Local variables are private to a particular automaton. Global variables in UPPAAL can be accessed by all the automata in the model, i.e. they are shared, and they play an important role in the communication between automata. This is because synchronization channels in UPPAAL are strictly for synchronization; it is not possible to pass data through a channel in UPPAAL.

The following table shows some global variables used in the model:

```
typedef struct{
        bool el[N];
} Set;
// GLOBAL State
Set MSETo, Joinable;
// Schedule for GM events: joins
Set Joining;
```

```
// Info sent in heartbeat messages
Set SAreqH;
// Info sent in the vote messages
meta Set Mset[N];
meta int[0,N] gsubV[N];
meta int[0,MaxGId] gidV[N];
```

In the GMP model there are two classes of global variables. The first class comprises variables that are updated only by the `Scheduler` and that are intended to control the behavior of the `Node` automata. Two variables of this class

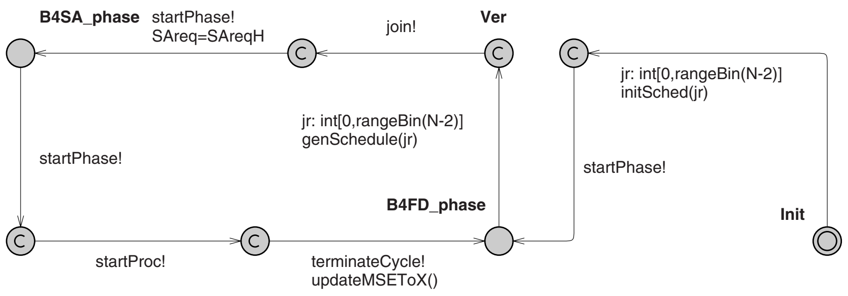


Fig. 3. Scheduler automaton for the model without faults

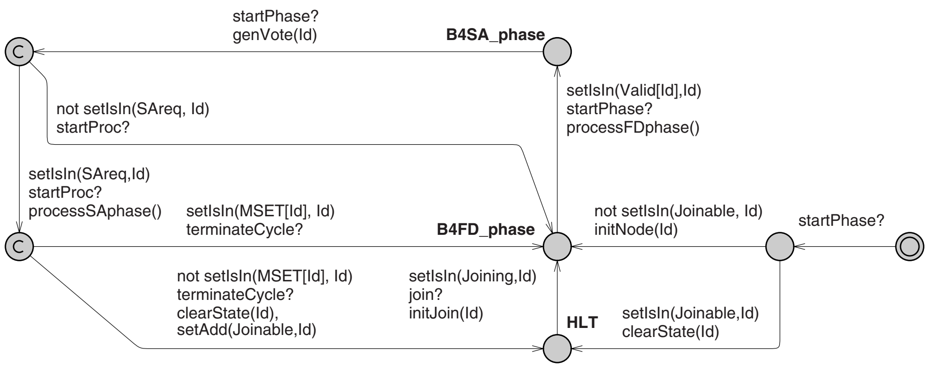


Fig. 4. Node automaton for the model without faults

are the sets `MSETo` and `Joining`. The former keeps track of the group membership as determined by an omniscient observer, whereas the latter keeps track of the nodes that try to join the group. The second class comprises variables that contain information that a node sends in the messages of the GMP. Two variables of this class are the set `SAreqH` and the array of sets `Mset`. The former contains the nodes that have requested to execute the SA-phase, i.e. each element of this array represents the SA-req bit of the heartbeat message sent by the corresponding node, whereas each element of the latter contains the group membership sent by each node in their vote message.

In addition to global variables the model comprises four broadcast synchronization channels: `join`, `startPhase`, `startProc` and `terminateCycle`. The latter two are not strictly necessary, but are used to eliminate intermediate states that are not relevant, thus reducing the size of the model's state space. (A detailed discussion on the use of synchronization is presented in Subsection 4.3.)

Automata. Figures 3 and 4 show the Scheduler and the Node automata for the basic model. In addition to the two phases of the GMP that are repeated one after the other indefinitely, the model includes an initialization phase. We briefly describe the base model considering each phase in turn. As stated above the Scheduler automaton controls the model by initiating the phases.

Model Initialization. This phase comprises the initialization of the variables of the model. It simplifies the verification of configurations with different number of nodes. In this phase, the Scheduler initializes itself and determines which nodes start as group members and those that do not.

Failure Detection Phase. Just before the FD-phase begins the `Scheduler` automaton is in location `B4FD_phase` and the `Node` automata are either in location `B4FD_phase` or in location `HLT`. Initiation of the FD-phase by the `Node` automata is controlled by the `Scheduler` automata.

At the beginning of the FD-phase, the `Scheduler` determines, with the help of a selection label, which of the nodes that can join the group will attempt it and initializes the `Joining` set with these nodes. After that it signals the selected nodes on broadcast channel `join`, so that they move from `HLT` to the `B4FD_phase` and therefore become ready to initiate the FD-phase. At this point, the nodes that will execute the FD-phase in this execution of the GMP are in the `B4FD_phase` state waiting on the `startPhase` broadcast channel.

Immediately after, the `Scheduler` signals on that channel, triggering the execution of the FD-phase by the nodes. The *actions* taken in this transition are specified inside function `processFDphase()`, which is executed by the `Node` automata and does the processing of the FD-phase of the GMP. In this processing, each `Node` automaton updates its `Mset` variable and the `SAreq` variable, as described in Annex A. In addition to its local state, each `Node` uses all messages it received in the FD-phase as input to `processFDphase()`. This information can be found by looking up sets `MSETo`, `Joining` and `SAreqH`.

Set Agreement Phase. The pattern of the set agreement phase (SA-phase) is very similar to that of the other phases.

Before executing the SA-phase the `Scheduler` and the `Node` automata that execute the GMP are all in their `B4SA_phase` location. Execution of the SA-phase by the `Node` automata is driven by the `Scheduler`.

When the `Scheduler` takes the transition out of location `B4SA_phase`, it signals on the `startPhase` broadcast channel, unblocking all the `Node` automata executing the GMP in this cycle. At this point each `Node` sends its vote, if any, by executing function `genVote()`.

Sending of a vote consists in updating global meta variables `gsubV` and `gidV`, with the values of the corresponding local variables, as described in the protocol shown in Annex A. The value of the `Mset` sent in the vote message, can be found directly in meta array variable `Mset`.

After that step, the `Scheduler` signals on broadcast channel `startProc` triggering the processing of the SA-phase. If a node is not in `SAreq`, i.e. the node has not observed any event that might lead to the change of the group membership, then it does not send any message in this phase and ignores all messages sent by other nodes, moving directly to location `B4FD_phase`. On the other hand, nodes in SAreq, must process the votes they receive. This is done in function processSAphase(), which implements the processing of the SA-phase of the GMP, described in Annex A. In this processing, a Node uses its own state variables, such as the MSet and the MSET sets, the meta variables with the votes sent, the SAreq set, which indicates which votes were actually sent, and the global variable Joining, which indicates which of those votes were sent by nodes joining the group.

In the absence of faults, this function is executed only when a node requests to join the group, and the outcome is the update of the state variables associated with the group, namely MSET, gid and gsub. However, in the presence of faults, a node may find out that its view of the group membership is different from that of the majority, or even that it is not able to determine the view of the majority. Under our fault assumptions, both cases indicate the occurrence of a fault in the Node and the protocol determines that the node must halt. To allow testing of this outcome, a node removes itself from its MSET if it must halt.

Thus, when the Scheduler signals on the terminateCycle broadcast channel, if a node is not a member of its MSET it moves to location HLT and is added to the set of nodes that can join the group(Joinable). Otherwise, the node moves to state B4FD_phase and becomes ready to execute the FD-phase again.

3.2 Modeling of Faults

In the previous section we have presented an UPPAAL model for the GMP in the absence of faults. In this subsection we describe how we model faults. The basic idea is to use *fault schedules* for each phase of the GMP execution. These fault schedules specify the fault events, i.e. send faults and receive faults, that each node will experience in the corresponding phase.

Generation of the fault schedules is done at two levels. At a system-wide level, the Scheduler determines which nodes have send faults and which nodes have receive faults. At a local level, each Node designated to have receive faults generates its own receive faults, i.e. determines on which messages it will experience a receive fault. Generation of the global fault schedule by the Scheduler makes it easier to ensure that the GMP fault assumptions are not violated. On the other hand, the generation of local receive fault schedule by nodes leads to a more structured approach and makes it easier to change the receive fault assignment policy.

Fault schedules are implemented as sets. The following state variables were added with that purpose:

```
// Global state variables
Set Faulty;
Set TxFaults;
Set RxFaults;
// Per node state variables - Scheduler needs to access them
Set NFaultsFD[N];     // Faults in the FD-phase
Set NFaultsSA[N];     // Faults in the SA-phase
```

In order to generate all fault schedules of interest in a compact way, we use select labels. The random integers generated by these labels are used either as

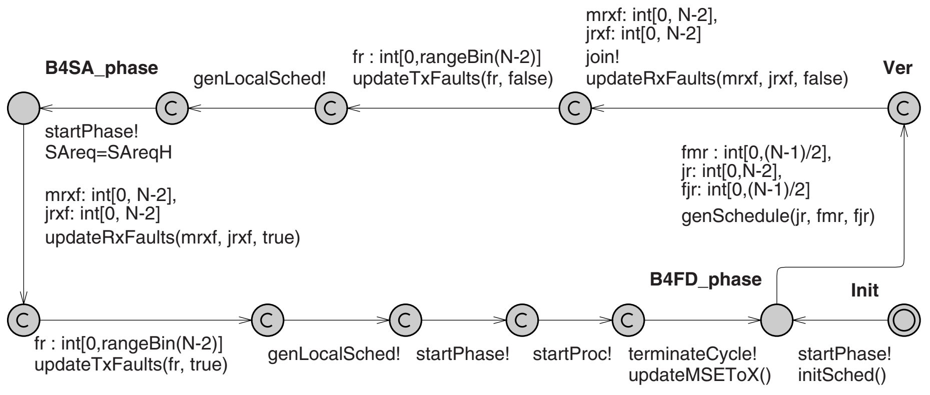


Fig. 5. Scheduler automaton for the model with faults

the number of nodes that fail, or as an encoding, with one bit per element, of a set of nodes that fail.

Figure 5 shows the `Scheduler` automaton that generates the fault schedules as described above.

To generate the global fault schedule, the `Scheduler` automaton determines which nodes fail in a GMP execution, at the beginning of each execution. I.e.,

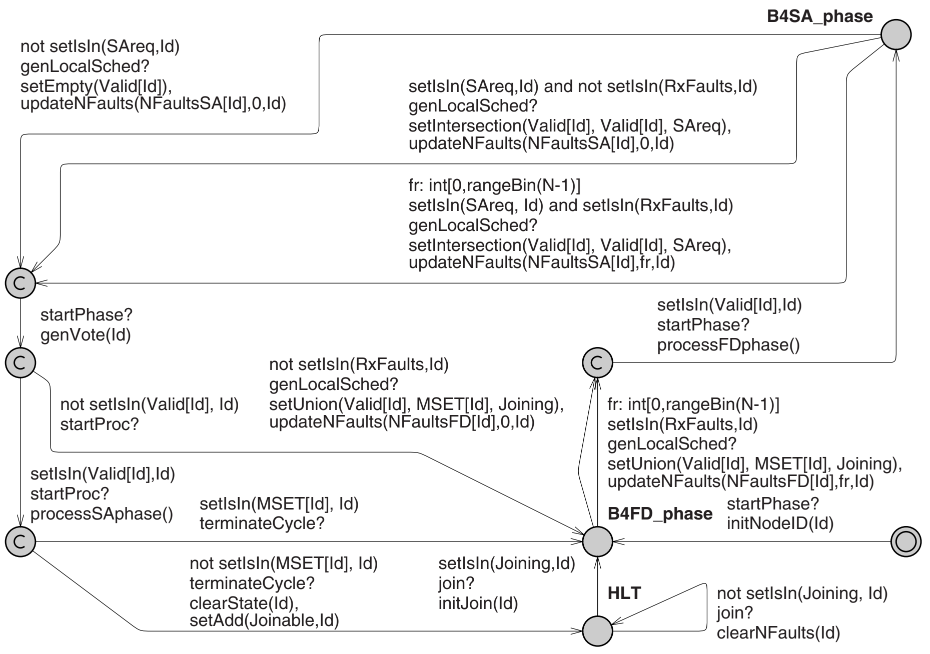


Fig. 6. Node automaton for the model with faults

now function `genSchedule()` not only determines which nodes will attempt to join the group, but also which nodes may fail. Then, before starting each phase, the `Scheduler` selects which nodes experience send faults and which nodes may experience receive faults.

Figure 6 shows the new `Node` automaton. Like in the `Scheduler` automaton, the structural changes concern only the generation of local schedules at the beginning of each phase. In addition, we had to make some changes to both `processFDphase()` and `processSAphase()`, because faults will affect which messages are received, and consequently processed, by each node.

We terminate our description of the modeling of faults with a reference to crash-faults, a kind of fault the GMP is supposed to tolerate but that we have ignored so far. It turns out that the model we have developed for receive and send faults subsumes the case of crash-faults. A crash-fault is a fault in which a node enters a halting state and takes no further action. To the other nodes an execution with such a fault is equivalent to an execution in which a node does not send any message, from some instant onwards. This behavior can be exhibited by this model, indeed a node that has send and receive faults from some point of its execution onward, moves to the `HLT` state and stays there indefinitely behaves like a crashed node.

4 Limiting the Size of the State Space

Modeling of faults makes the model inherently more complex. For example, in our fault model we consider that a node may fail in one of three ways: by crashing, by omitting to send a message or by omitting to receive a message. Given that each GMP execution has 2 phases that are not identical, each node may fail in 25 different ways. (Actually, this number is a lower bound as it considers only whether or not a node experiences at least one receive fault in a phase, disregarding the number of receive faults and on which messages these faults occur.)

In principle, one might argue that the number of receive faults in each phase is irrelevant and, in addition, that it does not matter in which phase of the GMP execution one node has a given fault. It turns out that none of these observations hold for the GMP, as some subtle fault scenarios that we described in [3] illustrate. We call these scenarios *masked faults*, as they correspond to cases in which a receive fault of one node is masked by another fault in the same or in the subsequent cycle. We have identified the following 3 cases:

1. SF_n in FD-phase; $RF_{m,n}$ in SA-phase.
2. $RF_{m,n}$ in SA-phase; SF_n in FD-phase.
3. $RF_{n,o}$ in FD-phase; $RF_{m,n}$ in SA-phase.

where SF_n means a send fault in node n, and $RF_{m,n}$ means the receive fault in node m on a message sent by node n. Thus, in the first two cases, the receive fault in m is effectively masked by a send fault in n, and therefore node m is not removed from the group. In the third case, the receive fault by m is masked by n's receive fault, and therefore node m is not removed from the group.

It is clear that if, e.g. in case 1 or 2, node m had a receive fault on all the messages sent, no masked fault would occur. This is because, for that to happen, all senders would have to fail, but such a fault scenario violates the fault assumptions of the GMP. It is also clear that if, e.g. in case 3, node m had its fault in the same phase as node n, then it would detected as faulty by the good nodes. These examples show that general principles [9] for model checking fault tolerant systems must be applied with care.

Still we can apply some general techniques to reduce the size of the state space. This is particularly important for model-checking the GMP because the state kept by each node is relatively large and we want to verify the protocol for configurations with a sufficiently large number of nodes to exhibit interesting behavior. We have found the following three techniques particularly useful in reducing the size of the state space of the model: 1) symmetry reduction; 2) priorities; 3) synchronization.

4.1 Symmetry Reduction

This technique is particularly effective for distributed algorithms, such as the GMP, where a set of identical components executes the same algorithm. Essentially, the idea is to take advantage of the fact that, for the GMP, it is not relevant which nodes are members or which of those are faulty, but rather how many nodes are group members or how many of those are faulty.

Uppaal itself provides support for symmetry reduction through *scalarset types*. They provide a way to tell the model checker about symmetries. Scalarset types can be seen as a bounded integer type with restricted operations, namely assignment and equality testing. Scalars may also be used as indices of arrays. Because of these restrictions, we found no clean way to model the GMP without using arrays indexed by scalarsets and whose elements contain scalarsets. However, for models with arrays indexed by scalarsets that contain elements of scalarsets the algorithm used by Uppaal for symmetry reduction is unlikely to provide any benefit [10]. Some preliminary experiments with simplified models with patterns of usage of scalarsets that would allow to model the GMP confirmed that. We have therefore implemented symmetry reduction directly in the model.

As stated above, for the GMP what is important is the number of nodes that fail, and not which nodes fail. Therefore, to eliminate "redundant states", the fault schedules are generated such that faults are assigned to nodes with higher identifiers. For example, in a configuration of 5 nodes, N0 to N4, in an execution where N4 is in location HLT and the remaining nodes are members of the group, the GMP tolerates one additional fault. In that event, which is generated randomly, the fault will be always assigned to node N3. This eliminates states where each of the remaining members fail instead of N3. Note that this technique does not eliminate all the redundant states. E.g., if instead of node N4 the node in HLT were node N3, in the event of a fault, that fault will be assigned to node N4. Although, such a state is equivalent to the state above, basically it can obtained by swapping the states of N3 and N4, our model is not able to eliminate it.

However, the number of these redundant states can be reduced, by adopting a consistent policy to select the nodes that join: the model generates randomly the number of nodes that will join, and then selects those that can join with lower identifiers. This policy, together with the one described in the previous paragraph, makes it highly probable that the group is composed by the members with lower identifiers, and that only nodes with higher identifiers will fail. In particular, it ensures that nodes N0 and N1 will never fail, whatever the number of nodes in the system, because the GMP requires at least two nodes, and the generation of faults in the model is such that it does not violate the GMP fault assumptions.

Selection of the faulty nodes that experience receive faults follows the same approach as that of the assignment of faults. For example, if nodes N4 and N5 are both selected as faulty, and the model determines randomly that one of them will have a receive fault in the FD-phase, then node N5 will be selected. On the other hand, selection of faulty nodes that experience transmission faults is done in a completely random way using select labels with a range from 0 to $2^{(N-2)} - 1$. The number selected is then used as an encoding of a set of N-2 elements and the latter is intersected with the set of faulty nodes. The reason for generating transmission faults in a completely random way is to allow all relevant combinations of send and receive faults. This approach is particularly effective for N smaller than 7, i.e. for at most 2 faulty nodes, in that it generates only 2 redundant pairs of receive faulty and send faulty node sets, in a total of 17, but the effectiveness of this policy decreases as the N increases.

Finally, we have also tried to explore symmetry reduction in the local receive fault schedules of nodes that are supposed to experience receive faults. Rather than generate completely random fault schedules, the receive fault schedules are only random with respect to messages sent by faulty nodes. With respect to messages sent by non-faulty nodes, we ensure that nodes will loose only the message sent by N0, which is guaranteed to be always a group member as explained above. This policy has two additional benefits. First, it ensures that all faulty nodes "collude" to remove a non-faulty node. Second, it does not eliminate fault schedules with multiple and reciprocal faults that may lead to subtle protocol behaviors. Again, this approach is particularly effective for N smaller than 7, in that it prevents redundant states, but for larger values of N redundant states will be generated.

4.2 Priorities

Another well-known technique to reduce the state space size of the model is to remove uninteresting interleavings. For example, in the GMP the order in which nodes execute the processing pertaining to each phase is not relevant. I.e., it does not matter whether node 0 executes before node 1 or the other way around. UPPAAL allows reducing these interleavings by means of *process priorities*. Using this feature, one can specify the order by which automata will take transitions when more than one transition is enabled at the same time, essentially inhibiting the transitions of automata with lower priority.

4.3 Synchronization

However, the use of priorities does not remove all the intermediate states. For example, considering that the higher the `Id` of a node the higher its priority, although node 1 will always take a transition before node 0, if both of them have enabled transitions, the intermediate state that occurs after node 1 taking its transition and before node 0 takes its transition will still be considered. One technique to remove these uninteresting states is to add synchronization, as we have done with the `startProc` and the `terminateCycle` broadcast channels. By adding the additional synchronization, all nodes take the transition simultaneously, and none of the otherwise intermediate states will be considered (unless it occurs in some other way).

It should be noted that although removing intermediate states is interesting for the sake of reducing the size of the state space, it may have adverse effects on the time for model checking. For example, we might reduce the size of the state space for about 30% for 5 nodes, by generating the schedules for send faults and receive faults on the same transition in `Scheduler`. However, verification of the properties described in the next section with such a model takes more than twice the time. The reason is that although the number of states is smaller, the number of transitions in the model is much larger, and therefore UPPAAL spends a lot of time testing transitions that in the end lead to the same state.

5 Correctness Properties

In [3], we have stated the Group Membership Problem in terms of the set of group members (M-SET) maintained by every node, and specified two properties:

Agreement: All non-faulty group members compute the same M-SET.
Validity:

1. A faulty node will be removed from the M-SET of a non-faulty group member in a bounded time interval;
2. A non-faulty node attempting to be reintegrated will be added to the M-SET of a non-faulty group member in a bounded time interval.

And we have also stated that the GMP ensured a bound of two TDMA cycles for removing a faulty member and one TDMA cycle for a non-faulty node to be reintegrated. The latter bound considers that the delay is measured starting on the instant the node sends a joining request.

UPPAAL allows the specification of the properties that a model must satisfy in a simplified version of CTL [6]. In particular it allows to specify safety properties like Agreement and Validity, using the $A\Box$ modal operator as follows:

Agreement: `A[] Sched.B4FD_phase imply Agreement()`
Validity1: `A[] Sched.B4FD_phase forall(i: int[0,N-1]) FDdelay[i]<3`
Validity2: `A[] Sched.B4FD_phase imply Validity2()`

where `Agreement()`, `Validity2()` are predicates that check the corresponding properties, and are as follows:

```
bool Agreement() {                    bool Validity2() {
    return                                return
        forall (i: int[0,N-1])                forall(i: int[0,N-1])
            ((setIsIn(MSETo, i)                   setIsIn(Joining, i) imply
              and not setIsIn(MSEToF,i))              ( setIsIn(MSETo, i)
                  imply MSET[i]==MSETo);                or not setIsEmpty(NFaultsFD[i])
}                                                       or not setIsEmpty(NFaultsSA[i]));
                                      }
```

Essentially, these expressions state that the corresponding properties hold after every execution of the GMP.

Actually, both Validity properties are bounded liveness properties and could have been checked using the leads to operator ($\rightsquigarrow$), also supported by UPPAAL. However, we found it more efficient to augment the model with some state variables and with the appropriate code. This augmentation concerned only Validity1. In particular, we added array `FDdelay` of integer variables that counts the number of GMP executions it takes for good members to remove faulty members from the group.

6 Verification Results

We verified both Agreement and Validity for configurations with three, four and five nodes. Table 1 shows the number of states stored and visited, as well as the time taken in checking each of the properties presented in the previous paragraph. For the case of 5 nodes, we present also the results we have obtained using an option provided by UPPAAL that reduces the memory requirements by not storing committed states, i.e. states in which at least one automaton is in a committed location. For the latter case, the table shows both the number of states stored and the number of states explored. When no memory reduction technique is used, only one value is shown because both numbers are equal.

The figures clearly show that the state space size increases exponentially with the number of nodes in the system, in spite of our efforts to explore symmetry at the level of the model. Although, the use of UPPAAL's memory reduction option allowed us to reduce the memory requirements for about one order of magnitude, the verification of these properties for configurations with more than 5 nodes leads to an exhaustion of memory resources.

Table 1. State Space Size (in thousand states) and approximate time execution for the different models and properties verified

Model		Agreement		Validity1		Validity2	
No.Nodes	Mem.Red.	No.States	Time(s)	No.States	Time(s)	No.States	Time(s)
3	N	5.3	0.5	5.3	0.5	5.6	0.6
4	N	220	56	220	56	221	57
5	N	14,237	28,920	14,232	29,640	14,870	30,060
5	Y (stored)	1,367	51,780	1,367	56,200	1,389	54,000
	Y (explored)	67,324		67,276		69,094	

Nevertheless, to be able to check the GMP for 5 nodes gives us a high confidence level in its correctness, because with 5 nodes we are able to generate rather subtle fault scenarios, such as the masked faults, that arise with the simultaneous fault of two nodes, which may be either members of the group or attempting to join. Although checking the correctness of the GMP for 7 nodes would provide an even higher confidence, because with that many nodes we could consider scenarios with 3 simultaneous faults, we believe that the change from 2 to 3 nodes does not lead to very different fault scenarios. Furthermore, to be able to verify the correctness of the GMP for a higher number of nodes in Uppaal is likely to require the use of *abstraction*, another well known technique of addressing the state space explosion problem.

However, the use of abstraction usually leads to models that are significantly different from the system being checked and consequently the level of confidence will be lower than if a model like the one we have developed were used. Indeed, our model includes an implementation of the GMP, except for the use of communication primitives such as `send` or `receive`, i.e. we abstract the communications layer. Given that Uppaal uses a syntax very close to C, it is straightforward to convert that model to a C implementation of the protocol.

Including an implementation of the GMP in the model allowed us to find a *bug* in the implementation outlined in [3] that is related to the fact that the number of group identifiers in an implementation must be bounded. In the GMP, shown in Annex A, the group id is incremented in step 9 of the SA-phase. At the level of abstraction of the specification, we considered that this variable is unbounded. However, in an implementation, as well as in model-checking, this variable has to be bounded. In [3], we have argued that an integer with a range from 0 to 3 is enough, and stated that the GMP did not require any other change. Although we were right with respect to the minimal range of group ids, we were wrong with respect to the need to change the GMP. The problem is in steps 1 to 3 of the SA-phase, where the maximum group id is determined and is then used to compute the majority set. With bounded group ids, these must be recycled, and therefore an id of 0 may be larger than an id of 3. Thus determining the maximum id is not straightforward, especially because joining nodes always send votes with a group id of 0, and faulty nodes may send any value, if they do not execute the SA-phase a number of times. The group ids of joining nodes can be easily fixed by ignoring the group id sent in their votes. The group ids sent by faulty nodes can be filtered by group members, taking into account the state of the GMP. However, joining nodes lack this state, and may compute a wrong group id. This may lead to an erroneous computation of the majority set in step 3. One way to fix this problem is to change the GMP so that joining nodes check that the majority they compute is consistent with the votes received (every node must consider itself a group member, and non-joining group members must agree on the group id). If it is not, they will cycle through all the group ids until a consistent majority is found, or they have tried all the ids. In the latter case, the joining node will consider itself faulty, and will halt.

7 Conclusion

We presented a formal verification of GMP, a protocol designed to provide a Group Membership Service for FlexRay, a minimalist middleware for the development of safety critical applications, that is likely to become the *de facto* standard bus for automotive applications.

The results obtained show that the GMP satisfies its specification for configurations of up to 5 nodes, providing us further assurance on its correctness. The fact that the model developed includes an implementation of the GMP contributes significantly to our confidence in its correctness, but also limits the number of nodes of the configurations that we are able to check. However, we strongly believe that we did the right choice, as it allowed us to detect a bug in the outline of an implementation we proposed in [3]. The alternative would be to use abstraction, which might lead to a model far removed from the GMP, and a doubt of whether the abstraction used was correct would always linger. Now that we have made a rather careful evaluation of the protocol correctness for configurations of up to 5 nodes, we plan to develop models based on abstraction to model check configurations with a larger number of nodes.

References

1. Rushby, J.M.: Bus Architectures for Safety-Critical Embedded Systems. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 306–323. Springer, Heidelberg (2001)
2. Makowitz, R., Temple, C.: FlexRay - A Communication Network for Automotive Control Systems. In: WFCS. 6th IEEE International Workshop on Factory Communication Systems (2006)
3. Rosset, V., Souto, P., Vasques, F.: A Group Membership Protocol for Communication Systems with both Static and Dynamic Scheduling. In: WFCS. 6th IEEE International Workshop on Factory Communication Systems, Torino, Italy, pp. 28–30 (2006)
4. Ip, C., Dill, D.: Better Verification through Symmetry. In: International Conference on Computer Hardware Description Languages, pp. 87–100 (April 1993)
5. Schiper, S., Toueg, A.: From Set Membership to Group Membership: A Separation of Concerns. IEEE Transactions on Dependable and Secure Computing 3(1), 2–12 (2006)
6. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In: Alur, R., Sontag, E.D., Henzinger, T.A. (eds.) Hybrid Systems III. LNCS, vol. 1066, pp. 232–243. Springer, Heidelberg (1996)
7. Yovine, S.: Model Checking Timed Automata. In: Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems, pp. 114–152. Springer, London, UK (1998)
8. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems. LNCS, vol. 3185, Springer, Heidelberg (2004)
9. Bernadeschi, C., Fantechi, A., Gnesi, S.: Model checking fault tolerant systems. Software Testing, Verification and Reliability 12, 251–275 (2002)

10. Hendriks, M., Behrmann, G., Larsen, K.G., Niebert, P., Vandraager, F.: Adding Symmetry Reduction to UPPAAL. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, Springer, Heidelberg (2004)

A GMP Protocol

The following is the GMP. It is specified considering the round execution model, commonly adopted in synchronous systems. I.e. each node begins its execution in its start state and then repeatedly executes, in lock-step with the other nodes, a round that comprises:

Communication step, in which each node generates a message, if any, that depends on the node's state, broadcasts it on the network, and receives the messages broadcasted in this step by all the nodes.

Processing step, in which each node generates the new state, by processing the messages received in the communication step of that round.

Group Membership Protocol

```
State
    P the set of all processors
    M-SET the set of group members, initially set to P
    M-set the set of candidate group members, initially set to P
    $u$ upper bound of the group's size, initially set to |P|
    group-id integer with group id, initially set to 0
    SA-req boolean indicating whether execution of the SA-phase should be per-
        formed, initially set to false
FD-phase
    Communication step:
        If processor is group member
        Then broadcast hearbeat message,
                with SA-req determined in the previous SA-phase
        Else if wishing to join group
        Then broadcast a join-req messsage.
    Processing step:
        1. Remove from the M-set every processor from which no heartbeat message
           was received.
        2. For every join-req message received
           add its sender to the M-set.
        3. If received a message with SA-req set
              or modified the M-set in 1 or 2
           Then set SA-req.
SA-phase
    If SA-req is set, then
    Communication step:
        Broadcast message with the M-set, the group's size upper bound, $u$, and the
        group-id.
    Processing step:
```

1. Let max-id be the maximum of the group ids received.
2. If the processor is joining
   Then set the group-id to max-id
   Else if its group-id is different from max-id
   Then halt
3. Let Maj-set be the result of applying the majSet function to P, the set of all the M-set's received from non-joining processors with a group-id equal to max-id, and to the minimum of all u's received in the same messages.
4. If the Maj-set
   (a) is undefined, or
   (b) is different from the M-set the processor broadcasted and the processor is a member of the group, or
   (c) is not a subset of the M-set the processor broadcasted and the processor is joining the group, or
   (d) does not contain the processor
   then halt.
5. Remove from the M-set
   (a) every group member from which an M-set different from the Maj-set was received;
   (b) every joining processor whose M-set is not a superset of the Maj-set;
6. Set u to the size of the M-set.
7. Remove from the M-set every processor from which no message was received in this phase.
8. If removed some processor from M-set in 7
   Then set the SA-req
   Else reset the SA-req.
9. Set the M-SET to the M-set and increment the group-id.

majSet function

```
Set majSet(Set S, SetofSet R, int n)
begin
    Set M to the ∅
    for every p in S do
        if p is an element of ⌈n/2⌉ or more sets in R
        then add p to M
        else if p is not an element of ⌈n/2⌉ or more sets in R
        then continue
        else return undefined
        end
    end
    return M
end
```

Revisiting Certification-Based Replicated Database Recovery*

M.I. Ruiz-Fuertes[1], J. Pla-Civera[1], J.E. Armendáriz-Iñigo[2], J.R. González de Mendívil[2], and F.D. Muñoz-Escoí[1]

[1] Instituto Tecnológico de Informática, 46022 Valencia, Spain
{miruifue,jpla,fmunyoz}@iti.upv.es
[2] Universidad Pública de Navarra, 31006 Pamplona, Spain
{enrique.armendariz,mendivil}@unavarra.es

Abstract. Certification-based database replication protocols are a good means for supporting transactions with the *snapshot* isolation level. Such kind of replication protocol does not demand readset propagation and allows the usage of a symmetric algorithm for terminating transactions, thus eliminating the need of a final voting phase. Recovery mechanisms especially adapted for certification-based replication protocols have not been thoroughly studied in previous works. In this paper we propose two recovery techniques for this kind of replication protocols and analyze their performance. The first technique consists in dividing the recovery in two stages, reducing the certification load and the amount of information to be recovered in the second stage. The second technique scans and compacts the set of items to transfer, sending only the latest version of each item. We show that these techniques can be easily combined, reducing thus the recovery time.

1 Introduction

Data replication is a technique used to increase the fault tolerance, availability and performance of distributed systems. In the context of database replication it is well known that eager update-everywhere techniques are the best approach when consistency and performance are the goals [1]. Recently, some performance studies [2] have shown that certification-based replication protocols provide the best performance, although at the cost of a non-negligible abortion rate with heavy loads due to their optimistic nature.

If these replication protocols are implemented in a middleware architecture, the resulting system becomes easily portable to different DBMSs. Additionally, it does not depend on the current version of a specific DBMS; i.e., new releases of the same DBMS could be used as the underlying persistence layer without needing any change in the middleware code.

But all this machinery is useless without a complementary recovery strategy adequately tailored to the data replication protocols being used. It is worth noting that a database, either replicated or not, holds a lot of persistent state and its recovery is quite

* This work has been partially supported by the EU FEDER and Spanish MEC under grants TIN2006-14738-C02 and BES-2007-17362.

difficult. In the non-replicated case, periodical backups or checkpoints are used. In the replicated case, the same approaches could be used as a starting point if no activity were allowed during the failure interval, but such solution is immediately discarded if availability is an aim. Another easy solution consists in transferring the full database to the recovering replica, but this raises problems since the amount of information to be transferred would be huge. So, the regular approach consists in remembering which was the state of the faulty replica before crashing, transferring only its missed updates. Despite this, if the failure interval is long, once again the full database transfer could be a valid option [3].

So, recovery is an important problem in database replication, which has been mostly overlooked so far. Studying efficient ways of doing recovery is an important step toward a complete solution to middleware-based database replication.

Although there are some recovery works for replicated databases [3,4,5,6,7,8], none of them has presented a performance study of the proposed solution, nor a comparison with previous works. The aim of this paper is to tailor two general solutions to the particular case of a certification-based replication protocol, and to study the recovery time of such proposals in a real database replication middleware system. Additionally, these two recovery techniques try to optimize two different steps of the recovery process and can be easily combined in a single recovery protocol, providing a highly reduced recovery time (at least, when compared with the other single techniques presented in this paper).

So, the contributions of this paper are: (a) to adapt some recovery techniques to certification-based replication protocols, (b) to propose a new recovery technique that combines two different optimizations, (c) to analyze the performance of different recovery protocols using a simple benchmark, and (d) to show that our new recovery proposal is able to improve the results of the other compared approaches.

The rest of this paper is structured as follows. Section 2 describes the assumed system model. Section 3 explains certification-based replication protocols, and how they manage historic information that can be also used for helping the recovery tasks. Section 4 describes the recovery strategies that will be benchmarked in this paper. Later, Section 5 presents the performance study, which is followed by an analysis of related work in Section 6, and a final Section 7 where the conclusions can be found.

2 System Model

We assume a partially synchronous distributed system where message propagation time is unknown but bounded. Such system is composed by N nodes and each one of them holds a complete copy of a given database. So, full replication is being assumed.

Database replicas may fail and recover, according to the *crash-recovery with partial amnesia* failure model [9]. Note that once a transaction has been committed, the underlying DBMS guarantees its persistence, but on-going transactions are lost when a replica fails. This provides a *partial amnesia* effect.

The recovery solutions explained in the following sections assume that the underlying database provides the *snapshot* [10] isolation level (SI, for short). This generates a GSI (Generalized SI) level [11] when a *certification-based* [2] replication protocol is

```
Initialization:
  1. lastvalidated_tid := 0
  2. lastcommitted_tid := 0
  3. ws_list := $\emptyset$
  4. tocommit_queue_k := $\emptyset$
I. Upon operation request for $T_i$ from local client
  1. If select, update, insert, delete
    a. if first operation of $T_i$
      - $T_i$.start := lastcommitted_tid
      - $T_i$.priority := 0
    b. execute operation at $R_k$ and return to client
  2. else    /* commit */
    a. $T_i$.WS := getwriteset($T_{ik}$) from local $R_k$
    b. if $T_i$.WS = $\emptyset$, then commit and return
    c. $T_i$.priority := 1
    d. multicast $T_i$ using total order
II. Upon receiving $T_i$ in total order
  1. obtain wsmutex
  2. if $\exists$ $T_j \in$ ws_list : $T_i$.start $<$ $T_j$.end $\wedge$
      $T_i$.WS $\cap$ $T_j$.WS $\neq \emptyset$
    a. release wsmutex
    b. if $T_i$ is local then abort $T_i$ at $R_k$ else discard
  3. else
    a. $T_i$.end := ++lastvalidated_tid
    b. append $T_i$ to ws_list and tocommit_queue_k
    c. release wsmutex
III. $T_i$ := head(tocommit_queue_k)
  1. if $T_i$ is remote at $R_k$
    a. begin $T_{ik}$ at $R_k$
    b. apply $T_i$.WS to $R_k$
    c. $\forall$ $T_j$ : $T_j$ is local in $R_k$ $\wedge$ $T_j$.WS $\cap$ $T_i$.WS $\neq \emptyset$
        $\wedge$ $T_j$ has not arrived to step II
        (this is analyzed by our conflict detector,
        concurrently with the previous step III.1.b)
        - abort $T_j$
  2. commit $T_{ik}$ at $R_k$
  3. ++lastcommitted_tid
  4. remove $T_i$ from tocommit_queue_k
```

Fig. 1. SIR-SBD algorithm at replica R_k

used. Our recovery strategies could also be extended to stricter isolation levels, like the *serializable* one –or even to more relaxed ones, like *read committed* [12]–, but supporting such stricter levels with a certification-based replication protocol requires readset propagation, and there are better replication approaches for those levels; e.g., the *weak-voting* [2] one.

In order to implement a certification-based replication strategy, a *group communication system* [13] is being assumed, providing a safe total-order multicast, and *same view delivery* [13]. If network partitions arise, the system uses a *primary component* model; i.e., a single isolated component (or subgroup) is able to progress, if and only if it holds a majority of the preconfigured system nodes.

3 Certification-Based Replication Protocols

There have been many replication protocols supporting the SI level. Most of them have used a *certification-based* replication architecture. In this paper, we take as the basis for our recovery approaches the SIR-SBD (certification-based) replication protocol already presented in [14] (See Figure 1). Certification approaches follow these rules (for the SI level):

1. Each transaction is initially executed in a single replica: the delegated [2] one.
2. When the transaction requests its commitment (for simplicity, we do not consider aborted transactions), its writeset is locally collected and sent to all replicas using a reliable total-order broadcast with safe delivery [13]. This is shown in step I.2 in Fig. 1.
3. All replicas maintain a log of delivered writesets (variable ws_list in our protocol), and using a given validation technique (step II.2 in Fig. 1), are able to certificate such incoming transaction. If the validation succeeds, the incoming writeset is added to the log, and the transaction is committed. If not, the writeset is discarded and the transaction is aborted in all replicas (indeed, only its delegate replica needs to roll-back it).

This validation/certification process is completely symmetrical in all replicas, since all replicas have identical copies of the same *writeset log*. No additional message is needed for deciding whether each transaction should be committed or aborted.

This also makes possible that transactions use a global identifier in all replicas. For instance, in our sample replication protocol the end logical timestamp is used as such identifier, and only the committed transactions receive such an ID (step II.3.a).

The writeset log being used in each replica for certification purposes should be pruned in order to prevent an endless size increase. In practice, the oldest writeset can be removed from such log as soon as the local node knows that every node has successfully applied such writeset. Note that the successful application of a writeset implies that all local conflicting transactions –i.e., those that might block such writeset application– should be aborted, since otherwise a deadlock might arise –if such blocking local transactions are blocked in subsequent accesses–, and the writeset application gets uselessly blocked. Anyway, once a certified writeset has been applied in a given replica, it is guaranteed that no local conflicting transaction that would have been certified against such writeset exists in such replica (since the underlying DBMS concurrency control ensures this). Aborting such local conflicting transactions as soon as possible increases the performance of certification-based replication protocols, as it has been shown in [14].

However, when a node fails it can not apply any subsequent writesets and this might prevent log trimming. On the other hand, this fact can be taken as the basis for designing a basic recovery protocol, since the information to be provided to the faulty node when it recovers is precisely the one already contained in the writeset log. Note however that the writeset log can also be pruned in these cases, but this leads to the adoption of another recovery strategy, using other mechanisms to find out which is the information to be transferred to the recovering replica.

4 Recovery Strategies

There have been several works [3,5] that propose general recovery approaches for replicated databases. This section describes a basic recovery approach, and complements it with two optimizations, showing that both optimizations can be easily combined and that such combination provides the best results (at least, the best results among all approaches being considered here).

4.1 Basic Recovery

The basic recovery approach consists in exploiting the writeset log being used for certification purposes. Once at least one of the replica nodes has crashed, the writeset log is not pruned and the *missed writesets* (i.e., those that have not been applied by the crashed replica) can be taken from this log when the failed node recovers. In practice, this is almost equivalent to propagating the missed messages to the recovering replica. A little difference exists: those writesets belonging to transactions that finally aborted are not propagated, since they were not stored in the writeset log. So, the recovering node does not need to repeat the whole validation process, and as a result of this, the work to be done is less than in the case of a regular replica that performed all the certification steps in order to manage each incoming writeset.

The detailed recovery procedure is the following: a recovering replica R_i joins the group, triggering a view change. As part of this procedure, the recovering protocol instance running in R_i multicasts an *ask-for-help* message indicating the $version_i$ of its last applied writeset –this version corresponds to the commit timestamp of the last transaction applied in that node. No message activity in the recovering node is done –all messages delivered are ignored– until this message is delivered. At this moment, the recovering node starts to enqueue the total order delivered messages –with writeset information about other transactions in the system sent by the rest of the replicas– to be processed later.

In parallel to this process, a deterministic procedure takes place to choose a recoverer re pli ca. The recoverer replica (R_j), after receiving the *ask-for-help* message, starts a recovery thread that sends a point-to-point message with all the missed writesets starting from $version_i + 1$, i.e., the recoverer node sends the portion of its `ws_list` that covers from $version_i + 1$ to the end of the `ws_list` at that moment.

When this point-to-point recovery message is delivered to the recovering replica, it stores this information in both the `ws_list` and the `tocommit` queue, as all these writesets were already certified in the recoverer node. Then, the recovering replica is ready to directly apply in the database the writesets in the `tocommit` queue and to start certifying its own enqueued total order messages –delivered after the *ask-for-help* message. Note that the certification of the enqueued messages must wait for the recovering information to be stored in the `ws_list`, as this structure is used in the certification process, but it is not necessary to wait for the application of these missed writesets in the database. In other words, just after the storage of the transmitted writesets in both data structures, the recovering node can act as in normal mode.

It is worth to note that the transmitted writesets are all applied in the context of a single transaction. This principle is maintained in the rest of recovery proposals, since it reduces the recovery time. On the other hand, if a second failure arises during the recovery, all the recovery process can be lost (but, hopefully, this is not the common case).

Such applied missed writesets can be pruned from the writeset log of all other replicas (if no more crashed replicas exist). Note that the recovering replica can start immediately new local transactions, but such transactions can not be certified until the whole sequence of missed writesets is transferred from the recoverer replica and appropriately inserted in the recovering writeset log.

This approach only makes sense for short-term outages; i.e., when the number of writesets missed by the recovering replica is small, independently of the real time length of such crash. Once the number of missed writesets exceeds a given threshold, such logged writesets needed only for recovery should be written to disk and read from there when the recovery is being done; i.e., they do not need to be maintained in main memory.

4.2 Two-Stage Recovery

The first proposed optimization on the basic recovery strategy described in the previous section consists in delaying the acceptance of the newly received writesets in the recovering replica; i.e., the writesets being delivered after the *ask-for-help* message in the *view* [13] V_k, that is the first one that re-includes the recovering replica (let's say, replica R_j). We refer to such writesets as *pending writesets*. Instead of delivering such pending writesets, and holding them until all missed writesets have been appended to the log, the pending writesets are momentarily discarded. A second stage is initiated when the recovering replica sends a message to the recoverer communicating that it has successfully applied all missed writesets. Then, the recoverer sends the successfully certified writesets that were discarded by the recovering replica.

Using this second transfer round has several advantages. Firstly, instead of holding and certifying all pending writesets, the recovering replica receives them already certified. Secondly, the amount of writesets that should be enqueued waiting for the certification process in the recovering replica is reduced. Moreover, all second-stage pending writesets are applied in a single transaction.

4.3 Compacting Recovery

The second optimization consists in compacting the sequence of missed writesets that was used in the basic recovery strategy. To this end, only a single version (the latest one) of each item being found in the original sequence of missed writesets is held in the compacted writeset list. Instead of transferring the full missed writeset sequence, only the compacted list is sent to the recovering replica.

Its appropriateness for SI certification-based replication protocols is easily justifiable [11]. Note that SI needs the start and end timestamps for each certifying transaction and the end timestamp for all writesets stored in the log in order to complete the certification process. Thus, when a SI transaction T_i is being certified, it gets aborted if any logged writeset W_k has an end timestamp in the [start,end] interval of T_i and T_i's writeset and W_k have a non-empty intersection. The compacted writeset list can be expanded if the original writeset timestamp was stored for each of its data items. It is worth noting that this generates a writeset log that does not contain the complete original information, but all items that have been removed from such rebuilt writeset log still maintain their newest version in the list; i.e., only the "old" repetitions of a given item have been removed, and its newest instance was still present in the compacted log. Since all transactions that need to be certified in the recovering replica have a [start,end] interval that terminates with an end timestamp that is trivially newer –since such transactions have multicast their writesets once the missed writeset sequence was computed and compacted in the recoverer– than any of the original missed writesets, there is no problem in discarding older repetitions of the compacted items.

The main advantages of this second optimization are the following ones. First, the amount of information to be transferred from the recoverer replica and applied in the recovering one can be reduced. Second, and as in the basic strategy, the recovering replica can start new local transactions immediately and they still have a non-negligible probability of success if they do not access any of the items updated during the failure interval. Third, and most important, this optimization is compatible with the one discussed in Section 4.2 and each one complements the other, as will be seen in the next subsection.

4.4 Combined Recovery

This last strategy combines the two previous optimizations. The two-stage solution discussed in Section 4.2 reduces the amount of writesets being processed by the recovering replica in the second recovery stage, whilst the compacting approach presented in Section 4.3 reduces the size of the writeset list to be transferred in the first (or single, if both optimizations are not combined) stage. So, there is no incompatibility between both approaches, and once combined they are able to further reduce the recovery time of any of such strategies, as shown in Section 5.

5 Performance Evaluation

In this section we intend to measure several aspects of our recovery strategies. Firstly, how long does it take the recovery depending on different parameters: crash interval, overall load, and new local transactions service in the recovering replica. Secondly, the impact of each optimization on the recovery time. Thus, Section 5.2 shows the results for short crash intervals varying the load and allowing as soon as possible new local transactions in the recovering replica. Section 5.3 repeats part of the latter experiments with the heaviest load and preventing new transactions from starting in the recovering replica. Finally, Section 5.4 analyzes the results for long crashes and a medium overall load.

In these tests we have varied several parameters –system load, number of clients, amount of missed writesets, . . . –, but several others that are also important remain fixed: e.g., number of nodes, and performance of the group communication system. It is worth noting that variations in these latter parameters would provide also variations in system load that also varies the number of missed writesets in a given crash interval. So, the effects of such unvaried parameters can be simulated varying the other ones.

Note also that these experiments have chosen a non-favorable test-case for the recovery strategies described above. Firstly, the test load consists only of read-write transactions; i.e., there are no read-only transactions, since they are not significant for the recovery analysis. As a result, with an apparent light load, the system is almost overloaded. Note, however, that the recovery time would be the same with a load five times greater and an –quite common in some kinds of applications– 80% of read-only transactions. Secondly, the duration of the crash intervals is not very long, and this is unfavorable for the compacting optimization since the longer the crash interval, the greater number of times items have been updated by the alive replicas; i.e., the compacting ratio would also be greater. Thirdly, these tests have generated an abortion rate below 5% in all cases, and this reduces a lot the benefits of the two-stage recovery strategy (Note that with a bigger abortion rate, the amount of non-transferred writesets in the second stage would have also been bigger). Our aim is to show that even in these unfavorable scenarios, the tested approaches can improve the results of the basic recovery strategy.

Perhaps a detailed analysis varying the abortion rate would have been welcome, but certification protocols are optimistic and an abortion rate below 5% for a write-intensive load is difficult to achieve. Additionally, as already commented above, a bigger abortion rate would improve the results of both the two-stage and the combined recoveries.

These tests have been done using our MADIS [15] middleware. MADIS mechanism that collects transaction writesets is implemented using standard SQL constructs. Also, only standard SQL mechanisms are used in order to apply these writesets. This enhances portability as no particular modification is made in the DBMS core, but penalizes performance. Additionally, it uses also standard mechanisms to deal with writeset collision detection, as already described in [14]. Such mechanisms are able to slightly enhance the performance of middleware replication protocols.

We have used a MADIS cluster composed by 4 replica nodes. Each node has an AMD Athlon(tm) 64 Processor at 2.0 GHz with 2 GB of RAM running Linux Fedora Core 5 with PostgreSQL 8.1.4 and Sun Java 1.5.0. They are interconnected by a 1 Gbit/s Ethernet. In each replica, there is a varying number of concurrent clients (from 4 to 12).

Each client executes an endless stream of sequential transactions, each one accessing a fixed number of 20 random items for writing, with a fixed pause of 500 ms between each pair of consecutive transactions. Each test begins with the execution of 500 globally committed transactions, after that, a failure occurs in a random replica (the failure of a replica consists in killing its process). Clients previously served by the failed node are not redistributed among the rest of nodes. The failure lasts for a period in which a varying number of global transactions is executed by the other replicas. After this time, the failed node restarts and begins the recovery process until it reaches the state of any of the other replicas. The test continues once the recovery ends, until the commitment of 500 more transactions, and then the experiment finishes.

To accomplish the comparison, we use a database schema with a single table with two columns and 10,000 rows (400,000 rows in Section 5.4). One column is declared as primary key, containing natural numbers as its values. This is a very simple database schema, but the usage of a standard performance-related schema (e.g., that of TPC-C or TPC-W) will not significantly vary the obtained results, since the original transactions need not be re-executed in the recovery steps. Instead of this, only their writesets need to be applied and the time needed to accomplish such task does not depend on the original transaction SQL sentences, but on the amount of updated objects.

Each result depicted in Figures 3-5 shows three different curves. Two correspond to the recovering and recoverer replicas, showing how many transactions have been committed in each of them (Recall the last_committed_tid variable in Fig. 1). The third one corresponds to another alive replica; i.e., a non-faulty replica that does not collaborate in the recovering process. The aim of this third curve is to show whether the recovering tasks introduce a significant load in the recoverer replica. If so arises, its curve is below that of the other non-faulty replica until the recovery overload disappears.

To better illustrate the meaning of such curves, let us concentrate in a particular figure. For instance, the results shown in Fig. 3.1.a. Note that in such figure, one of the three plotted replicas crashes at time 46 seconds, once it already committed 510 transactions. Until this moment, all the replicas perform similarly so their curves overlap. Note that until then, the system was able to serve transactions at a rate of 510/46 TPS (i.e., 11.08 TPS). Later, such replica is restarted at time 86 seconds; i.e., the crash interval has lasted 40 seconds in this example. Then, such a recovering replica sends a message to the recoverer one in order to start the recovery process, and waits for the arrival of such sequence of missed writesets. The complete sequence is received and its application is finished at time 104 seconds. However at such time there is still a gap between the number of successfully committed transactions at the recovering replica and the number of committed transactions at other replicas. But the recovering replica is able to progressively reduce such gap and finally terminates its recovery at time 120, once 1,529 transactions have been committed in the whole system. Note that in this example with a minimal system load, the final performance has been better once the recovery terminated (1,529/120=12.74 TPS) than prior to the replica failure (11.08 TPS). Note also that in this example, the recovery process has needed 120-86=34 seconds in order to heal a crash that lasted 40 seconds. These time measures may seem too long, but it has to be considered that our MADIS system is not a commercial prototype, so general performance is not our main goal. This way, presented measures have to be observed in order to study relative improvements between different techniques whilst absolute values should not be considered.

In spite of this, it may seem that the transactions per second obtained at each phase of a test have an evolution slightly different from expected. In the graphs, the TPS achieved during a period of time is obtained as the slope of the curve in that period. From the beginning of each test, it can be seen that the slope grows progressively before the crash time as the clients are started in the system. In spite of the fact that the crash is done after 500 transactions, it still occurs before the system has reached its maximum performance –this would explain that the TPS after the complete recovery of the node is, in some cases, greater than that before the failure–. However, this should not have impact on the recovery time, that is, in fact, what we are analyzing in this paper.

5.1 Statistical Comments

When a performance analysis is made, usually the average values are shown in the plots, and each experiment is repeated many times, until the obtained standard deviation ensures that at least the 95% of the results are close enough to that mean. This approach is not appropriate for plotting the recovery graphs, since we are not only interested in the recovery time, but also in the trend followed by the recovering and recoverer replicas. Plotting the average number of committed transactions in the recovering and recoverer replica at any time will not provide easily readable graphs.

As a result, we have taken a single representative curve for each kind of experiment. In order to choose such graph, we have repeated each experiment at least fifty times and we have selected its median value. To show the appropriateness of such median values, Figure 2 plots the *box-and-whisker* diagrams in all test cases. These results will be explained in Sections 5.2 and 5.3. A diagram of this kind plots a box that starts with the first quartile value and ends with the third one. The median value is also depicted into the box. Finally, the "whiskers" are continuously plotted until the minimum or maximum value if such extreme values do not exceed 1.5 times the interquartile interval; otherwise, all the exceeding values are considered as *outliers* and plotted as dots.

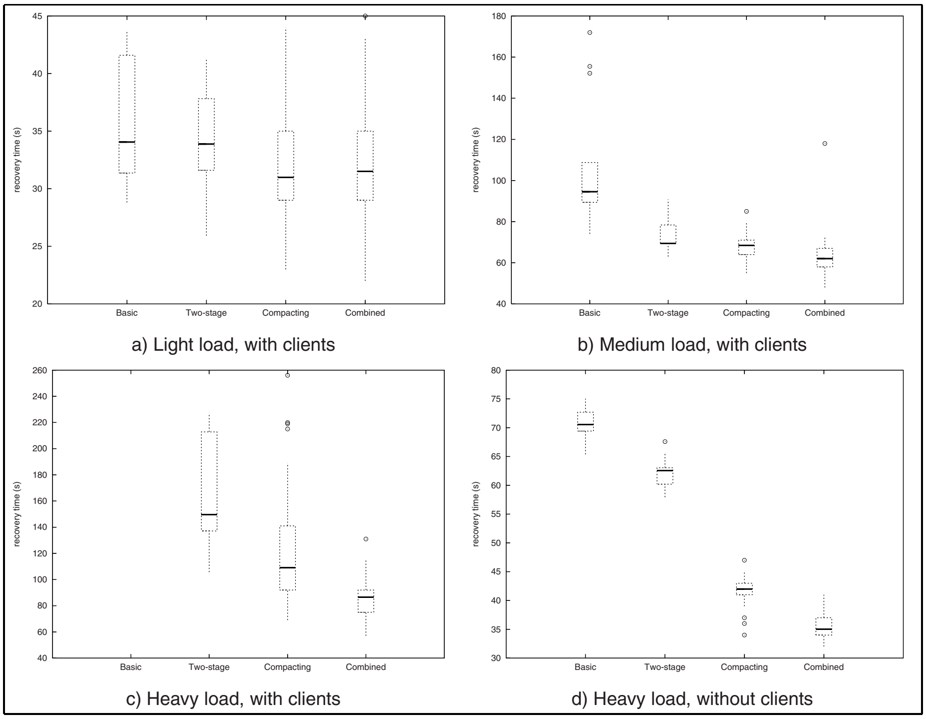


Fig. 2. Box-and-whisker plots for all tests

5.2 Evaluation with Clients

In this first set of tests, all recovery strategies accept new local transactions in the recovering node as soon as possible. Three different experiments have been designed, varying the system load and the crash interval. Their parameters are summarized in the following table:

	System load	**Crash length**
Light	4 cl/node	500 trans.
Medium	8 cl/node	1,000 trans.
Heavy	12 cl/node	2,000 trans.

Despite considering the third test as a “heavy” load and a “long” crash interval, it is worth noting that all these three cases can be managed holding the whole writeset log in memory. In practice all these test cases should be considered as examples of short outages.

Figure 3.1 shows the results for the basic recovery strategy. The three graphs obtained for the light, medium and heavy load experiments show that with a light load this strategy is not much worse than any of the other optimized strategies. However, with a medium or heavy load, things change a lot. Indeed, when transactions are immediately

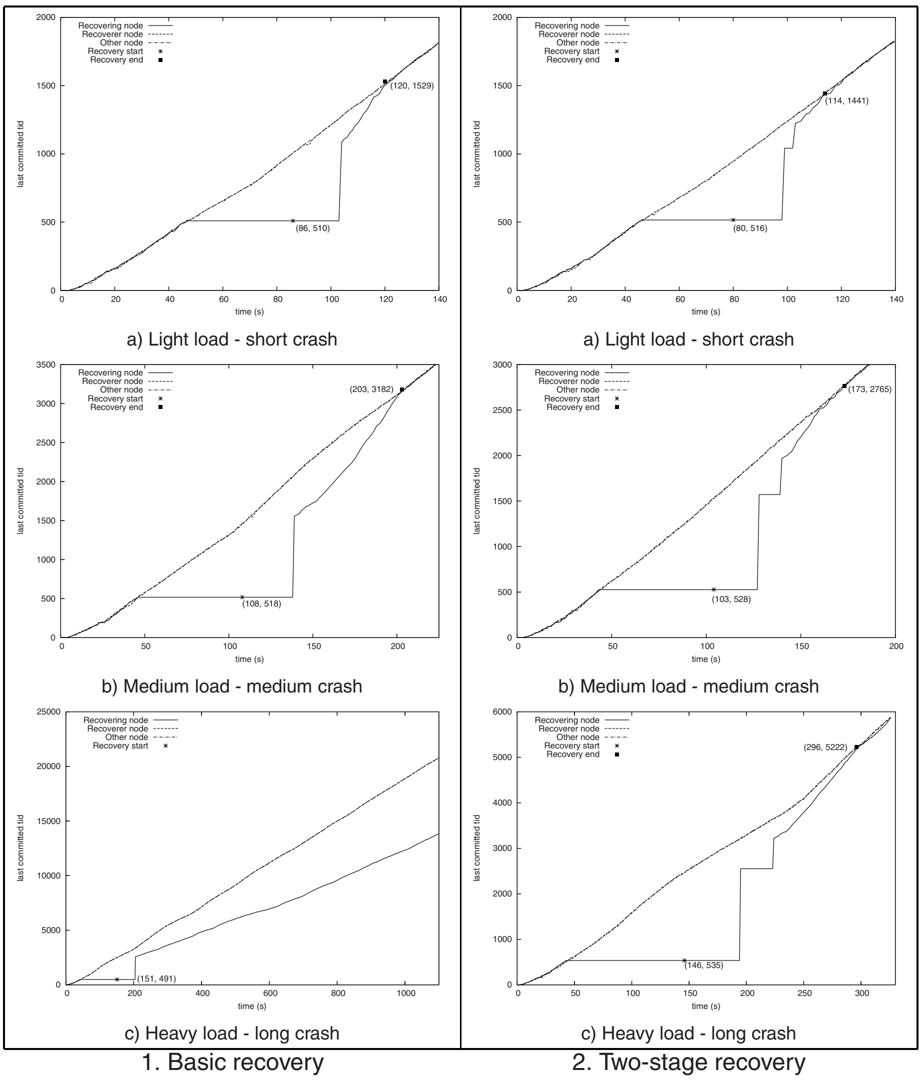


Fig. 3. Basic and two-stage recoveries

accepted in the recovering replica, it is not able to cope with its work in the heavy load case and can not complete its recovery, as shown in Figure 3.1.c. Note that such load almost overloads our regular replicas, and the recovering replica needs also to apply all its missed writesets. This also explains why no box was plotted for this recovery technique in Figure 2.c.

Figure 3.2 shows the results for the two-stage recovery strategy. It needs 34 seconds to complete the recovery in the light-load case, i.e., an identical median time to that needed by the basic approach. Note, however (see Figure 2), that its mean time is slightly lower (in the basic recovery, the second-to-third interquartile box is bigger

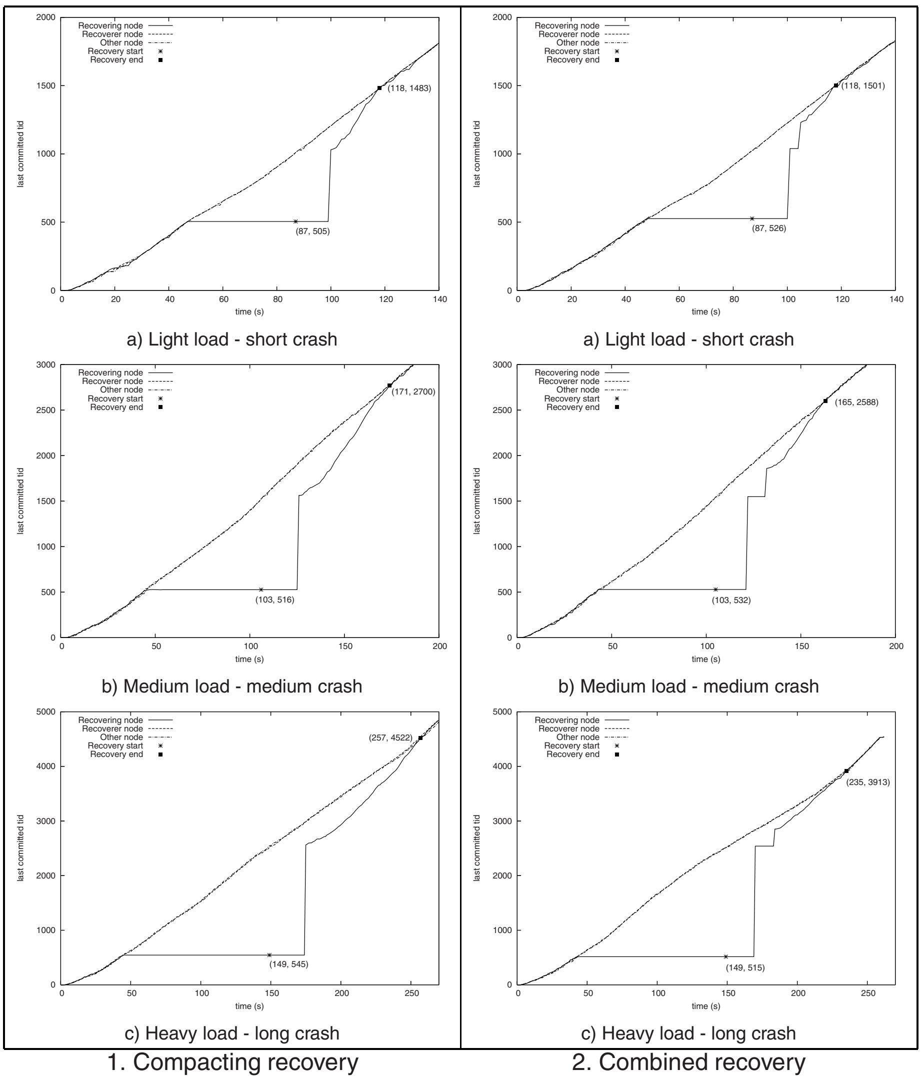


Fig. 4. Compacting and combined recoveries

than in the two-stage recovery, and this generates a greater average); i.e., the two-stage recovery is a bit better than the basic approach, even with light load. With medium load, this approach completes its recovery in 70 seconds, whilst the basic one needs 95 seconds; i.e., the two-stage one is 26.3% better. Finally, in the heavy-load case, this strategy is the first one able to recover and completes its work in 150 seconds.

Better results have been provided by the compacting recovery (Figure 4.1). In the light-load case, compacting recovery is able to terminate its tasks in 31 sec. (a 8.8% improvement, since it lasts 34 sec. with the two-stage strategy). In the intermediate load, it needs 68 sec., achieving only a 2.8% of improvement. Finally, in the heavy-load

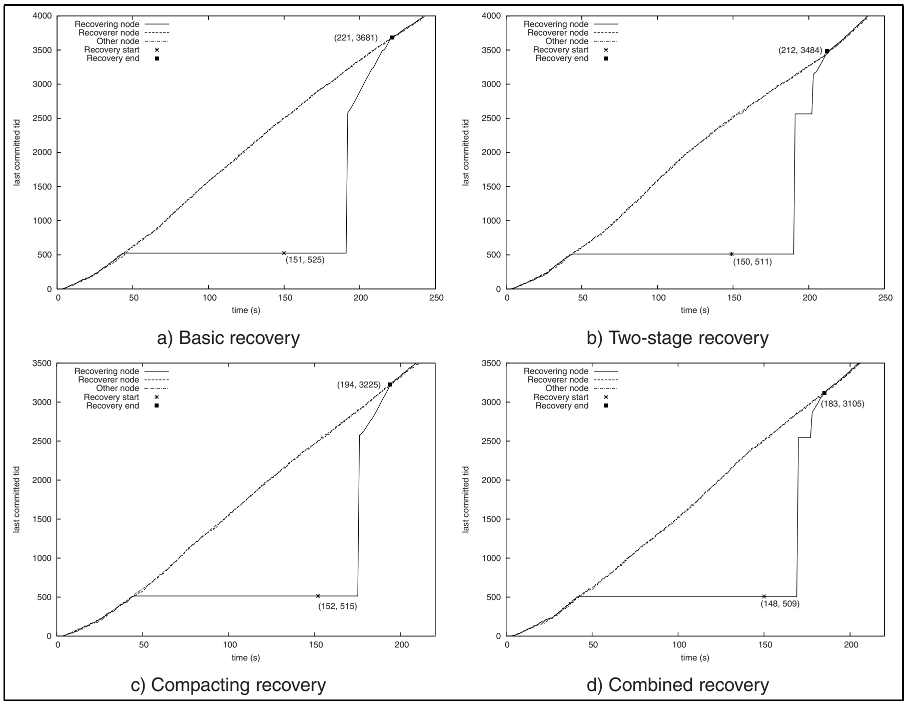


Fig. 5. Results without new local transactions in the recovering site

case needs 108 sec., 42 less than the two-stage approach (i.e., 28% better). Additionally, the overhead in the recoverer replica during the compacting step is negligible since no gap appears between the recoverer and *other-node* curves in Figure 4.1.

Finally, the combined recovery provides the best results (see Figure 4.2). It completes the recovery in 31, 62, and 86 seconds, for the light, medium, and heavy loads, respectively. This implies 0%, 8.8% and 20.3% respective improvement against the light, medium and heavy results in the best strategy previously analyzed (the compacting one). Note that the results get better when heavier loads are introduced. Indeed, in the light-load case, no benefit is observed.

5.3 Evaluation Without Clients

In this second kind of test, we analyze the behavior of the four recovery strategies preventing new local transactions from starting in the recovering replica. This reduces the load in the recovering replica, but delays a bit its service provision to the client applications. Nonetheless, the global effects to clients are not very important, since some of the transactions admitted in the previous tests were finally aborted in the recovering replica. For instance, with a heavy load and the combined recovery strategy, 75 transactions were admitted and terminated in the recovering replica during the recovery phase, and only 20 were committed.

The results of this second experiment are shown in Figure 5. In this case only the heavy-load case has been used, since it is the scenario where the differences among the strategies could be significant.

In such setting, the basic strategy is already able to cope with the recovery work and it provides non so bad results. It is able to terminate the recovery in 70 seconds. The two-stage recovery provides better results (it needs 62 seconds, being 11.4% better).

The compacting and combined strategies provide the best results. Thus, the compacting approach (shown in Figure 5.c) is able to terminate the recovery in 42 seconds (i.e., 40% better than the basic strategy, and 32.2% better than the two-stage one). Note that with this crash length, it is highly probable that many items were written several times and that the transferred information was much smaller than in the basic approach. Finally, the combined strategy still reduces a bit such recovery time and completes its tasks in 35 seconds, improving in a 16.6% the results of the compacting approach. Note that this final strategy is able to combine the best characteristics of both optimizations: fast application time of a reduced set of item values due to the compacting approach, and a pre-processing of the certification tasks in the second stage.

5.4 Long-Term Outages

Finally, we have also tested all recovery approaches with a long crash interval of 20,000 lost transactions on a 400,000-item database with 8 clients per replica (and with 4 replicas, as above) and without accepting transactions in the recovering replica. In these cases, the failure interval lasts around 200 minutes whilst the recovery time ranges between 966 and 1,418 seconds. We have analyzed also the impact of the abortion rate on the two-stage strategy. To this end, we have defined two different hot-spot sizes (1,000 and 10,000 items) and 40% of the transactions accessed to only items in such a hot-spot, whilst the other 60% uniformly accessed to the whole database.

Mean results are summarized in Table 5.4. In this case, each experiment has been repeated 30 times and the standard deviation was lower than 2% of the mean value. As it can be seen, the two-stage strategy depends a lot on the abortion rate. With an abortion rate of almost 1% (hot-spot of 10,000 items), its recovery time is almost 41% bigger than that of the compacting approach. Moreover, combining the two-stage with the compacting does not provide any benefit.

However, with an abortion rate close to 6% (small hot-spot) the recovery time is decreased in the two-stage case 36.5 seconds, whilst in the compacting case it only decreases 31.21 seconds. Additionally, combining both optimizations the overall recovery time is the lowest one among all the obtained results.

Table 1. Recovery times for long-term crashes (in seconds)

Hot-spot	Recovery Protocol			Abortion
size	2-stage	Compact.	Combined	rate
1,000	1,381.29	979.73	966.73	5.89%
10,000	1,417.79	1,010.94	1,018.55	0.94%

6 Related Work

Some papers have proposed recovery techniques for replicated databases assuming replication protocols using reliable broadcast mechanisms [3,4,5,16,6,17,18,8], but –up to our knowledge– only a few of them [16,6] have presented a performance study of their recovery time. Additionally, none of them is centered on certification-based replication protocols, the focus of this paper.

The two-stage strategy has been proposed for certification-based replication protocols providing the GSI [11] isolation level in [8] where only its correctness was outlined. Such strategy was inspired by some details of the recovery protocol proposed in [5] for a *primary copy* [2] replication protocol. But such solution was also an evolution of that initially presented as an N-stage proposal in the *lazy data transfer* algorithm of [3]. None of those three papers [3,5,8] measured the recovery time of such approaches.

The other two papers [16,6] that presented performance results of recovery protocols are not directly comparable with this one –they are not intended for certification-based replication protocols–. Irún [16] proposes a lazy recovery solution for a hybrid replication protocol that may be configured either with eager or lazy behavior. Lazy solutions usually have a bad impact on the abortion rate, but his solution overcomes such problem with an outdatedness estimation function. Castro [6] proposed also a combined mechanism that switches between a log-based recovery (similar to the basic recovery strategy discussed here) and a version-based one (similar to our compacting strategy) depending on the crash length. At a glance, it shares some of the characteristics of our combined approach, but there are important differences. Its solution should choose a given approach for achieving recovery, whilst ours is able to combine the two-stage and the compacting strategies at once. Its performance study was mainly oriented to prove that its combination of two different recovery strategies was able to provide the best results in a given range of crash lengths. Our performance study, besides doing something similar, also compares our combined solution with three other recovery strategies.

7 Conclusions

This paper has analyzed the performance –i.e., recovery time– of four different recovery strategies for certification-based database replication protocols. To this end, these four recovery approaches have been implemented in a middleware system and have been tested with different loads and crash intervals. The results show that a basic recovery approach can be easily optimized with two different techniques: two-stage recovery and compacting. Additionally, we propose a fourth strategy that combines these two compatible optimizations and show that it is able to further improve the recovery times of the other analyzed approaches. Up to our knowledge, this is the first paper that compares actual implementations of multiple recovery approaches for certification-based replication protocols.

References

1. Wiesmann, M., Schiper, A., Pedone, F., Kemme, B., Alonso, G.: Database replication techniques: A three parameter classification. In: SRDS, pp. 206–215 (2000)
2. Wiesmann, M., Schiper, A.: Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng. 17(4), 551–566 (2005)
3. Kemme, B., Bartoli, A., Babaoğlu, O.: Online reconfiguration in replicated databases based on group communication. In: DSN, Washington, DC, USA, pp. 117–130 (2001)
4. Holliday, J.: Replicated database recovery using multicast communication. In: NCA, pp. 104–107. IEEE Computer Society, Los Alamitos (2001)
5. Jiménez-Peris, R., Patiño-Martínez, M., Alonso, G.: Non-intrusive, parallel recovery of replicated data. In: SRDS, pp. 150–159 (October 2002)
6. Castro, F., Esparza, J., Ruiz, M.I., Irún, L., Decker, H., Muñoz, F.D.: CLOB: Communication support for efficient replicated database recovery. In: 13th Euromicro PDP, Lugano, Sw, pp. 314–321 (2005)
7. Armendáriz, J.E., Garitagoita, J.R.: Muñoz, F.D., de Mendívil, J.R.G.: MADIS-SI: A database replication protocol with easy recovery. Technical Report ITI-ITE-06/05, Instituto Tecnológico de Informática, Valencia, Spain (July 2006)
8. Armendáriz, J.E., Muñoz, F.D., Juárez, J.R., de Mendívil, J.R.G., Kemme, B.: A recovery protocol for middleware replicated databases providing GSI. In: ARES, Vienna, Austria (April 2007)
9. Cristian, F.: Understanding fault-tolerant distributed systems. Comm. ACM 34(2), 56–78 (1991)
10. Berenson, H., Bernstein, P.A., Gray, J., Melton, J., O'Neil, E.J., O'Neil, P.E.: A critique of ANSI SQL isolation levels. In: SIGMOD Conf., pp. 1–10. ACM Press, New York (1995)
11. Elnikety, S., Zwaenepoel, W., Pedone, F.: Database replication using generalized snapshot isolation. In: SRDS, Orlando, FL, USA, pp. 73–84 (October 2005)
12. Salinas, R., Bernabé, J.M., Armendáriz, J.E., Muñoz, F.D.: SIRC-Rep: A multiple isolation level protocol for middleware-based data replication. Technical Report ITI-ITE-07/03, Instituto Tecnológico de Informática, Valencia, Spain (February 2007)
13. Chockler, G., Keidar, I., Vitenberg, R.: Group communication specifications: a comprehensive study. ACM Computing Surveys 33(4), 427–469 (2001)
14. Muñoz, F.D., Pla, J., Ruiz, M.I., Irún, L., Decker, H., Armendáriz, J.E., de Mendívil, J.R.G.: Managing transaction conflicts in middleware-based database replication architectures. In: SRDS, Leeds, UK, pp. 401–410 (October 2006)
15. Irún, L., Decker, H., de Juan, R., Castro, F., Armendáriz, J.E.: MADIS: A slim middleware for database replication. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 349–359. Springer, Heidelberg (2005)
16. Irún, L., Castro, F., García, F., Calero, A.: Lazy recovery in a hybrid database replication protocol. In: XII JCSD, Ávila, Spain, pp. 295–307 (June 2004)
17. Castro, F., Irún, L., García, F., Muñoz, F.: FOBr: A version-based recovery protocol for replicated databases. In: 13th Euromicro PDP, Lugano, Sw, pp. 306–313 (2005)
18. Armendáriz, J.E., Muñoz, F.D., Decker, H., Juárez, J.R., de Mendívil, J.R.G.: A protocol for reconciling recovery and high-availability in replicated databases. In: 21st International Symposium on Computer Information Sciences, Springer, Heidelberg (2006)

A Survey of Fault Tolerant CORBA Systems

Muhammad Fahad[1], Aamer Nadeem[1], and Michael R. Lyu[2]

[1] Department of Computer Science
Mohammad Ali Jinnah University, Islamabad, Pakistan
mhd.fahad@gmail.com, aamern@acm.org
[2] Department of Computer Science and Engineering
Chinese University of Hong Kong, Hong Kong S.A.R., China
lyu@cse.cuhk.edu.hk

Abstract. CORBA is an OMG standard for distributed object computing; but despite being a standard and wide scale acceptance in the industry it lacks the ability to meet high demands of quality of service (QoS) required for building a reliable fault tolerant distributed system. To tackle these issues, in 2001, OMG incorporated fault tolerance mechanisms, QoS policies and services in its standard interfaces as mentioned in its Fault Tolerant CORBA (FT-CORBA) specification. FT-CORBA Architecture used the notion of object replication to provide reliable and fault tolerant services. In this paper, we surveyed the different approaches for building FT-CORBA based distributed systems with their merits and limitations. We gave an overview of FT-CORBA specification; its requirements and limitations, and FT-CORBA Architecture. We have also revised the existing categorization of FT-CORBA systems by incorporating a fourth approach, i.e., Reflective Approach, in the categorization taxonomy. A comparison between different types of replication and FT-CORBA based systems is conducted to achieve quick insight on their features.

Keywords: CORBA Middleware, Object Replication Styles, Fault Tolerant CORBA Specification, Fault Tolerant CORBA systems.

1 Introduction

Distributed systems are used in a variety of application domains in which services are provided by independent components working together as a single transparent system. In distributed systems, CORBA is accepted as a standard because of its inherent location transparency, portability, interoperability and language independence [1]. With these features, CORBA was made a standard for distributed object computing by the Object Management Group (OMG) [2]. In CORBA, Interface Definition Language (IDL) defines interfaces to objects. Clients have to implement IDL interfaces to access server functionality and this makes CORBA language independent. By location transparency, clients can invoke server objects without worrying about the location of the server objects. Portability makes CORBA independent of specific ORB and the system can be implemented and used on top of any CORBA-compliant ORB. This is achieved by the Portable Object Adaptor (POA), a component of CORBA, which is responsible for making server-side functionality appear as CORBA object to clients.

Interoperability of CORBA ensures the system to be used by the clients and servers, running on ORBs from different vendors. Despite these benefits, CORBA does not address partial failures and does not provide totally ordered multicast of messages while building distributed systems [2], which are the key factors of fault tolerance.

To provide fault tolerance in distributed CORBA based systems, Fault Tolerant (FT) CORBA specification defines interfaces, QoS policies, associated fault tolerance mechanisms and services to enhance the reliability of CORBA applications [3]. Existing fault tolerant CORBA systems provide fault tolerance through replication of CORBA objects. By replicated objects, fault tolerant services are provided even if one of individual entities fails. A replicated object is implemented by a set of distinct CORBA objects called an *object group,* i.e., an abstraction to provide replication transparency and failure transparency [4]. These systems differ mostly at the level at which the replication mechanism support is introduced. Felber and Narasimhan categorize the FT CORBA systems on this basis into three categorizes: *integration, interception and service* [1] and discuss the experiences and lessons learnt in building their two distinct FT CORBA systems.

This paper presents the overview of Fault Tolerant CORBA Specification; its architecture, requirements and limitations. We surveyed the FT-CORBA systems based on their FT properties and highlighted their prominent features and limitations. We analyzed the several different approaches that implement FT-CORBA and revised the existing categorization of FT-CORBA systems by incorporating a fourth approach called Reflective Approach in the existing categorization taxonomy. Moreover we compared the working of individual systems on different criteria and provide analysis matrix to achieve quick insight on their infrastructures.

The rest of this paper is organized as follows: Section 2 covers the approaches of building the fault tolerant CORBA system with their merits and limitations. Section 3 throws light on replication styles and comparison of these styles. Section 4 gives the overview of Fault Tolerant CORBA Specification; its requirement, architecture and limitations. Section 5 covers the Fault Tolerant CORBA systems with a critical look on their features. Section 6 shows our analysis about the FT-CORBA systems. Section 7 concludes the paper.

2 Fault Tolerant Approaches

According to the built-in support of replication logic, various fault tolerant systems are categorized into four approaches, which are termed as Integration approach, Interception approach, Service approach and Reflective approach. Taxonomies of the first three approaches can be found in [5,6,7]. With the passage of time new FT-CORBA Systems were built introducing fourth approach, i.e., Reflective approach. So here we incorporate the fourth approach in the existing (old) categorization taxonomy. Summarized features of these approaches are represented in Table 1.

2.1 Integration Approach

In this approach, support for replication is integrated transparently into the ORB. It integrates necessary fault tolerant replication by proprietary mechanisms in the ORB.

This is the most efficient approach but modification in the ORB makes this approach non-compliant with the CORBA standard, i.e., does not enable off-the-shelf ORBs to be used. In this approach, modified ORB gets a message from application objects, passes it to the adapter object that multicasts it by using underlying toolkit. Fault tolerance mechanisms and replication strategies are transparent to the client as these are integrated into the ORBs. Portability is not achieved but interoperability can be achieved depending on the support for *IIOP* invocations (Internet Inter-ORB Protocol, a CORBA Standard for invocations).

2.2 Interception Approach

In this approach, support for replication is provided underneath the ORB, which makes the replication logic transparent to the users. Messages from client and server are intercepted transparently, externally to the ORB by using low level (OS-level) interceptor and then multicast by the group communication toolkit. The use of low-level interceptor makes this approach non-portable. Moreover as there is no need of modification in ORB, thus systems built using this approach are ORB compliant. Interoperability can be achieved by writing an interception layer of each distinct OS.

2.3 Service Approach

In this approach, support for replication is provided through a collection of CORBA objects that reside above the ORB. As there is no need to modify the ORB, the systems built exploiting this approach are CORBA compliant, interoperable and portable. To use service objects that provide the policies and mechanisms for achieving fault tolerance, application objects require knowledge of these service objects and hence application code needs modifications to use their functionality. Each request from application objects to service objects passes through the underlying ORB, which increases performance overheads. Service objects are defined as IDL interfaces so they are independent of language constructs. Service objects can be made distributed by locating them on different hosts on the network.

Table 1. Comparison between Fault tolerant Approaches

System Features	Integration	Interception	Service	Reflective
ORB Compliance	No	Yes	Yes	Yes
Transparency	Yes	Yes	Depends on service implementation	Yes
OS dependence	No	Yes	No	No
Portability	No	Can be achieved	Yes	Yes
Interoperability	Depends on IIOP invocation	Yes	Yes	yes
Performance	Most efficient	Efficient	Good	Good
System Example	Electra, PPF, Orbix+Isis	Eternal, CARRIGE	OGS, DOORS, AQuA, Newtop, FTS, IRL, Aquarius	FRIENDS, FT-MOP

2.4 Reflective Approach

Reflection approach separates the concerns between the application and the fault tolerance mechanisms and enables off-the-shelf ORBs to be used. It employs metalevel architecture to integrate fault tolerance in CORBA systems and provides a means to develop transparent fault tolerance software as any CORBA software with different object-oriented languages. In this approach the replication necessarily involves creating a single point of failure outside the client's failure domain, thus partially defeating the purpose of the replication (no single failure is visible to the client). The use of *Metaobjects Protocol* (MOP) and restricted reflective features of some object-oriented languages makes this approach different from other approaches. Using this approach MOP can be implement as compile time or runtime. But the integration of runtime and compile-time MOPs enables more efficient functionality for fault tolerance. This MOP is CORBA compliant which enables the execution and the state evolution of CORBA objects to be controlled. Metaobjects are not only used for the purpose of fault tolerance but can also be used for security purposes. Interoperability can be achieved with the engagement of Metaobjects protocol.

All these approaches support different types of object replication styles in which they replicate their constituent objects. The need for object replication is to increase the reliability and performance of the system. Failure of a replica does not affect the services provided, as other replicas are there to give the required services. Performance issues arise when distributed systems need to scale in number and geographical area [4]. Fault tolerance benefits can be achieved only when object replication maintains strong replica consistency. Strong replica consistency means that all the replicated objects should have the same state and they perform the same behavior. There are many issues which should be analyzed while maintaining strong replica consistency [7,8]. First, all the replicas perform the same sequence of operations in the same order. Second, to perform a single invocation multi-replicated client objects initiate a request to replicated server objects, thus each of server objects receive multiple requests made by each of the client object. Therefore, the system should be capable enough to detect duplicate requests. Third, systems which support multithreading should carefully analyze different threads and the functions they perform. Fourth, in case of failure of replicated objects, recovery mechanisms should be transparently managed to provide the reliable fault tolerant services.

3 Replication Styles

The replication logic is a set of protocols, mechanisms and services that allow a CORBA system to handle object replication [9]. There are many styles of object replication but the main ones are Active replication and Passive replication [5,6,7]. Underlying mechanisms for both are the same but their role to provide strong replica consistency is different. Some of the FT-CORBA systems rely on proprietary group toolkit for replication logic implementation but others provide either centralized replication logic in its core or completely distributed above the ORBs [9]. A comparison between replication styles is represented in Table 2.

3.1 Active Replication

In Active Replication, all replicated objects are active and independently handle client requests and return the responses to the client. Duplicate responses should be detected and suppressed to provide client transparency. One of these active replica objects is called primary, while others act as backup. The crash failure of single primary is masked by the presence of other active replica by providing fault tolerant services; thus this style provides better fail-over time, and state transfer and recovery mechanisms are provided to regain the use of the crashed node. To ensure replica consistency, it consumes a lot of computational resources and totally ordered multicast of messages is needed to maintain the same state and to achieve same behavior by active replicas, i.e., it needs operations on the replicated objects to be deterministic. It shields fastest recovery from faults.

3.2 Passive Replication

In Passive Replication only one operational replica is active, termed as primary, to fulfill client request. It requires less memory and processing costs, and shields slower recovery from faults. On the basis of recovery mechanisms it has two variations:

Warm Passive. Only one server replica (primary) is active in each object group and remaining replicas are preloaded into the memory and are synchronized periodically to handle state transfer while crash faults. To achieve this state synchronization, totally ordered multicast as well as deterministic operations are needed. Only active replica is operational to fulfill client request, while backups are running for the sake of state storage and state transfer in case of primary failure. When primary fails, new primary is selected from the backup replicas.

Table 2. Comparison between Replication Styles

Analysis Parameters	Active	Warm Passive	Cold Passive
Number of operational replica	All	Only primaries	One
Fail-over time	Very low	Medium	Very high
Computational resources	High	Medium	Low
Duplicate message detection and suppression required	Yes	Yes	No
Totally ordered multicast required	Yes	Yes	No
Operations on replicated objects	Deterministic	Deterministic	Non-deterministic
Recovery from faults	Fastest, very Rapid	Rapid	Slower

Cold Passive. Only one server replica is active and the remaining replicas are not even preloaded into the memory. State of the primary is logged into the storage for recovery mechanisms. If the primary fails, new primary is created and state is transferred from logged storage to the new primary, which increases the fail-overtime. This approach uses less resources and non-deterministic operations, as only one replica is operational at a time.

4 Overview of FT-CORBA

In 1998, Object Management Group (OMG) felt the need of making fault tolerant standard properties for CORBA Architecture for adding availability and reliability in CORBA applications. Hence issued a Request For Proposal (RFP) that results the Fault Tolerant CORBA specifications in early 2000 [3]. FT-CORBA specification addressed the issues of entity redundancy, fault detection, and fault recovery. This section throws a light on FT-CORBA Specification.

4.1 FT-CORBA Architecture

The Fault Tolerant CORBA Architecture [3] is achieved by handling issues of object replication transparently, fault detection and recovery mechanisms in CORBA Architecture as shown in the Fig. 1. Major components with their functionality are:

Replication Manager. Replication Manager has three components; *Property Manager*, *Generic Factory*, and *Object Group Manager*. Property Manager allows application developer to choose and set object group properties i.e. replication style, consistency style, membership style etc according to requirements. Generic Factory creates objects and makes object groups. Object Group Manager adds or deletes members.

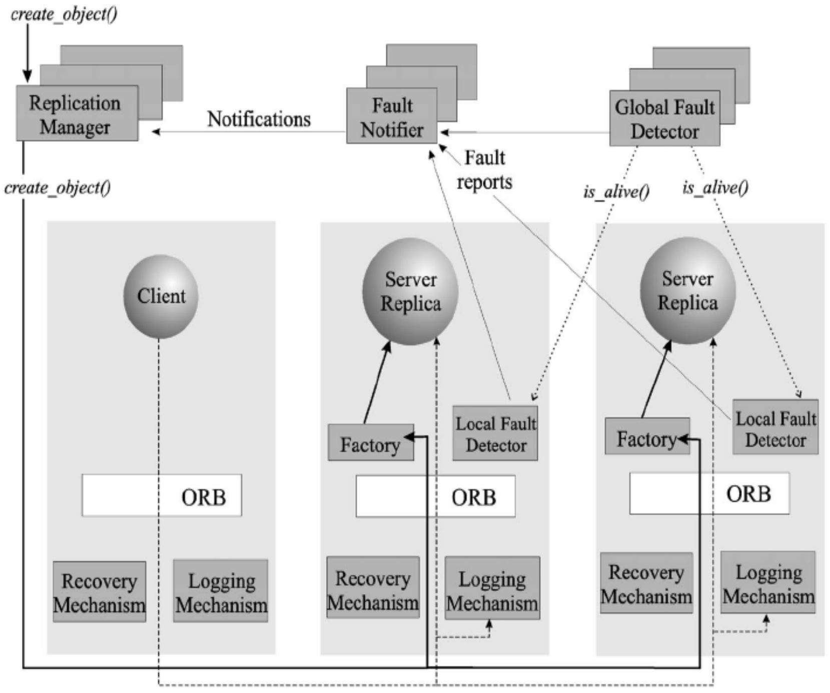


Fig. 1. The Architecture of Fault Tolerant CORBA [7]

Fault Detector and Fault Notifier. *Fault Detector* supports Pull Model and Push Model based fault monitoring [3]. In *Pull Monitoring*, crash faults are detected by invoking an isAlive() method of monitored object asking about its aliveness. If monitored object does not reply within some time interval then it is assumed that object has crashed. By this approach, application checks the status of objects when it is needed [10]. In *Push Monitoring*, crash faults are detected on the basis of I_am_Alive() messages sent by monitored object who tells about its aliveness. Crash fault of monitored object is assumed when it does not send message telling about its aliveness. By this approach fast detection of the crash failure is achieved [10]. Whenever a fault is detected, *Fault Detector* reports the fault to *Fault Notifier*, which diverts it to *Replication Manager* to take necessary actions. There should be separate *Fault Detector* and *Fault Notifier* components according to standard Fault tolerant CORBA specification.

Logging and Recovery. FT-CORBA defines a logging and recovery mechanisms by two IDL interfaces (*Checkpointable* and *Updateable*). The logging mechanism periodically stores object related information on the log, and recovery mechanism retrieves log information to restore valid state to the crashed replica.

4.2 Requirements of FT-CORBA Specification

According to FT-CORBA specification [3], system build on CORBA middleware should preserve CORBA object model for the infrastructure-controlled consistency style, and extended format of *Interoperable Object Reference* (IOR) should be used for the individual replicas so that legacy ORBs that does not support fault tolerance can invoked methods on ORBs that support fault tolerance and vice versa. Each component should be replicated to avoid single point of failure; moreover creation and deletion of objects, fault detection, and recovery mechanisms should be invisible to client to achieve transparency. In case of failure of replica, client's request should be transparently redirected to other available replica and client ORB systematically re-initiate the request until the request fulfills.

4.3 Limitations of FT-CORBA Specification

FT-CORBA specification [3] has many limitations that are: i) Clients running on non-FT-CORBA can invoke methods/operations on an object group, supported by the fault tolerant infrastructure without taking the benefits of its fault tolerant properties. ii) To achieve interoperability and full fault tolerance, the hosts with in a domain should use fault tolerant infrastructure and ORBs from the same vendor. iii) To achieve strong replica consistency, specification addressed that application objects should have deterministic behavior. iv) There is no support for partitioned systems, and *Network-Partitioning faults*, *Commission faults* (wrong results generated by the objects), and *Correlated faults* (Design Faults, and Programming Logic Errors) are not addressed in the specification.

5 Existing Fault Tolerant CORBA Systems

Many FT-CORBA systems were developed to address the issues of secure group based communication for embedding fault tolerance by the notion of object replication. The evolution of FT-CORBA systems starts with the integration of fault tolerant properties in the ORB, but later on different approaches were introduced to build replication for ease of use and customization purpose to provide fault tolerance in CORBA based distributed systems. The following sub-sections present the brief introduction to various Fault Tolerant Systems.

5.1 Electra

The Electra [4] is one of the earliest implementation of fault tolerant CORBA systems, developed at the University of Zurich which exploits the integration approach. It was the first time using the strengths of CORBA model and improving the weaknesses of CORBA model with group communication for consistent ordering of distributed events and transactions, handling of partial failures and support of asynchronous communication.

The first research based CORBA object request broker, Electra, combines the benefits of CORBA object model and virtual synchrony with reliable group communication as part of an ORB to achieve fault tolerance. As the replication logic is embedded into the ORB, it neither is ORB compliant nor maintains interoperability of CORBA architecture. Also we cannot achieve interoperability using Electra. The key focus of Electra is to enable ORB with build-in fault tolerant capabilities. All the special features of adding fault tolerance are enhanced by two C++ interfaces *Basic Adaptor Object* (BOA) Interface and *Environment Interface*, so C++ is the only target language for building fault tolerant CORBA based application using Electra prototype. Underlying toolkit, which is built on the model of virtual synchrony, provides reliable multicast. BOA provides active replication and Environment Interface is responsible for synchronous, asynchronous and deferred-synchronous communication. Adaptor object has the code specific to the toolkit so application developer can use another toolkit by simply relinking the application with the appropriate Adaptor Object. Basic Adaptor Object, which is hooked into the ORB, is responsible for replication services and mechanisms like creation and deletion of objects and object groups, and state transfer when primary replica fails. It also allows application developers to select the ordering protocol given by the toolkit according to requirements. Group communication is achieved by the subsystems (Horus, Isis) that are built on the model of virtual synchrony to maintain replica consistency.

5.2 Orbix+Isis

First commercially available Fault Tolerant CORBA system [11] developed by the IONA Technologies was Orbix+Isis, which exploits the integration approach. *Isis* developed by the Isis Distributed Systems was the first commercial toolkit built upon the model of virtual synchrony to provide high performance, totally ordered multicast and fault monitoring. *Orbix* is the C++ development environment to work on distributed CORBA objects.

It modifies ORB to use *Isis* toolkit which provides totally ordered multicast reliable communication, object groups and failure monitoring, whereas *Orbix* provides the object oriented environment to work on distributed objects and supports point-to-point communication. Fault tolerant replication mechanisms are implemented by using two base classes *ActiveReplica* and *Stream Event*. *ActiveReplica* provides transparent Active and hot-passive replication, and *Event Stream* (supports asynchronous requests using publish/subscribe paradigm) makes object groups and used for load balancing. *Orbix+Isis* allows application developers to select the object replication execution style. Transparent replications of server objects and filter mechanisms are provided by the Orbix specific smart proxies. Active replica execution style also gives an option to select the replication style. In Event Stream style, Event Streams are replicated which keep event history and Event Log. Servers registered to specific events are invoked by Event Stream when it receives the event from the client. Fault monitoring is based on two functions *_newMember()* and *_memberLeft()*. The former is called when an object joins group and latter one is called when the object leaves.

5.3 Eternal

Eternal [1,7], a FT CORBA standard, was developed at the University of California, Santa Barbara, which exploits the interception approach to provide transparent fault tolerance to ORB and application as well. It employs *Totem* toolkit for totally ordered multicast.

Eternal has an ORB compliant architecture but does not maintain interoperability of CORBA because when request came, it is captured by OS-level interceptor and then propagated to ORB, thus making Eternal OS dependant. Interoperability can still be achieved by writing a separate interception layer for every different ORB. It supports active and different types of passive replication (e.g. cold passive, warm passive) and logging-recovery mechanisms to provide reliable consistent replication. Active replication allows the Eternal to work, when single replica fails as this is masked by the presence of other active replicas and during recovery phase. For providing consistent replication it maintains three types of states; application level state, ORB/POA level state and Infrastructure state, and this distinguishes Eternal from other fault tolerant CORBA systems. It provides fault detection service based on user-defined timeouts to identify crash faults. It allows developers to select configuration management properties of fault tolerance, and employs mechanisms to overcome the non-determinism inherent in multithreaded CORBA applications.

5.4 DOORS

The Distributed Object Oriented Reliable Service (DOORS) [6] is an application-level framework developed at Lucent Technologies as an experimental middleware so that lessons learned during its implementation are integrated into the FT-CORBA standard. DOORS exploits the service approach to provide fault tolerance and follows an ORB compliant architecture which maintains interoperability of CORBA. The proposed architecture supports active and passive replication, but prototype implementation only provides passive replication. Both pull and push methods of fault monitoring are supported to provide fault detection and employs libraries for the transparent checkpointing of applications. Fault detection and fault notification are merged into fault detector component. It provides transparently fault detection and fail-over to the client. The prototype does not support recovery and logging mechanisms, and duplicate detection and suppression of messages for reliable fault tolerance. Replication Manager is responsible for configuration management and replication mechanisms by allowing application developer to choose and set object group properties i.e. replication style and consistency style, according to requirements. Fault Detector detects the faults and reports them to super fault detector that diverts them to replication manager to take necessary actions. There is no separate Fault Notifier component thus it violates the standard fault tolerant CORBA specification. All these component services act as CORBA objects above the ORB.

5.5 AQuA

The BBN Technologies and University of Illinois, developed the AQuA's gateway architecture to provide adaptive fault tolerance to CORBA systems [12]. Its architecture consists of Quality Objects, Proteus, Maestro/Ensemble and gateways. It replaces the ORB IIOP implementation with proprietary gateway which propagates IIOP calls to other CORBA objects by using *Maestro/Ensemble* toolkit. The *gateway* and the group toolkit employ the replication logic. Due to replacement of *only IIOP module* of ORB with gateway, it is regarded to exploit integration approach [9]. But as the *gateway* captures the initial request by client object which acts as an OS-level interceptor, it is regarded to exploit interception approach [6]. We classified AQuA exploiting service approach, as it provides replication via a collection of CORBA objects above the ORB [1]. Nevertheless, interoperability is achieved by implementing gateway for each different OS and ORB. The configuration management regarding fault tolerance properties can be set during runtime. Push-based or heartbeat fault monitoring is supported for fault detection. Different types of active and passive replication schemes are supported to tolerate crash and value faults. The application developer can set the level of dependability by *Quality Objects* according to desired application requirements and state of the distributed system during execution of the system. *Proteus*, a flexible infrastructure, has replicated dependability manager, gateway handler and object factory. The replication dependability manager makes decisions on reported faults, manages configuration properties, and replicas are created and deleted by the object factories.

5.6 FTS

FTS [13] as a lightweight CORBA fault tolerance service was developed at Israel Institute of Technology that maintains the portability and interoperability of CORBA ORBs. It aims to support transparent client-side replication and embeds fault tolerance in CORBA by utilizing the standard CORBA's *Portable Object Adaptor* (POA).

It provides fail-over transparency and reliable transparent fault tolerance by redirecting a client's requests during processing. It supports two types of fault detection; process-based which is monitoring of *Group Object Adaptor* (GOA) and object-based in which all the objects are monitored which are connected with GOA by push fault monitoring model. Active replication of server objects is supported. A set of components, which provide reliable functionality for fault tolerance, are merged into group object adaptor, which is built on the top of POA. *FTS Interceptors* detect faults during client-server replica communication and redirects a client's request to other replicas when they receive an indication of faults during request processing, thus adding reliable transparent fault tolerance to client applications. It partially supports network partitioning by imposing a primary component model.

5.7 IRL

IRL was developed by the University La Sapienza, Roma, Italy, which exploits the service approach [9]. It maintained the CORBA's interoperability and was built with supports of passive centralized replication logic. Later on, a distributed design was proposed to give more reliable fault tolerant properties [14].

With its replication logic implemented as CORBA objects above the ORB, IRL offers interoperable ORB compliant architecture in which all the components are deployed distributedly to avoid single point of failure, thus adding more reliable fault tolerant ways to handle client's request in a more transparent manner. Adding support of the client-side replication and the server-side replication to system makes IRL more reliable while achieving good performance. Object Replicas are distributed in different host domains for balancing loads and achieving high fault tolerance. To handle object creation and deletion, replication style and its management, *Object Group Handler* (OGH) and *Object Group* (OGs) Components were designed. Fault detector and fault notifier detect faults and provide fail-over transparency to clients. Host-specific IRL components as well as domain-specific IRL components handle failure management activities. Local failure detectors monitor crash faults by pull fault tolerant technique. Recovery mechanisms are carried out by Object Group component, which ensures strong replica consistency in a group.

5.8 OGS

Object Group Service (OGS) [5] developed at the Swiss Federal Institute of Technology, Lausanne, exploits the service approach as the first time in the history to provide fault tolerance in a CORBA system. It maintains interoperability and provides distributed replication support in building more reliable fault tolerance.

OGS supports a set of independent generic IDL specified interfaces, which provides transparent group invocations. It preserves portability of CORBA ORBs, and provides both reliable (for read-only client requests) and unreliable multicast of messages, and mechanisms for duplicate detection and suppression. Furthermore, it supports active and warm passive replication techniques, as well as fault monitoring by push and pull methods. *Group Service* component manages work related to objects and group membership and provides client transparency. The consensus service ensures the total ordered multicast and replica consistency, crash fault detection is done by monitoring service, and messaging service transmits client server invocations onto the transport layer. Replication service employs the user to select replication style and other fault tolerant properties. Clients implement IDL interfaces to use a set of services of known replicated server so it does not maintain replication transparency.

Furthermore, recovery services are used incase of failures of object replicas and for the transfer of application-level state.

5.9 Newtop

Newtop [15] was developed by the University of Newcastle, which exploits the service approach. It follows the similar approach as being implemented by OGS but it provides more group management facilities. It embeds the support for objects belonging to multiple groups and handling the failures due to partitioning. *Newtop Service Object* (NOS), provides the distributed mechanisms and handles client requests in a fault tolerant way. It achieves its functionality by three services implemented as an object, i.e., Group management service object, Invocation/multicast service object and Membership service object. Group management service is responsible for creation and deletion of objects from groups. Invocation/multicast service provides synchronous and asynchronous communication facilities and information about the object is kept by the Membership service. However it does not employ consistent remerging of the subgroups once communication is reestablished. Membership service is also held responsible for checking crash faults on the bases of a timeout protocol.

5.10 Aquarius

Aquarius was developed at the Hebrew University of Jerusalem, Israel [16]. It exploits the service approach and is based on Quorum Object Adaptor Architecture [17]. It provides the data-centric approach to build fault tolerance in CORBA.

Aquarius embeds server-side replication support by using the object adaptor approach like FTS. But it modifies the adaptor by adding an ordering protocol's algorithm. It employs proxies (stateless servers), which act as middle tier between client and server. These proxies propagate client requests to server and help to achieve efficient client-server invocation and transparency. It consists of two parallel threads of execution, one is responsible for propagating client requests to all replica servers and other is responsible for creating a total order of all client requests. Its architecture is similar to that of IRL but the middle tier of Aquarius uses independent entities that are not aware of each other and do not run any kind of distributed protocols among them. It applies the ordering protocol to maintain strong replica consistency. It utilizes RPC mechanisms that support asynchronous invocations for delivery of client requests to all replicas.

5.11 Pluggable Protocol Framework (PPF)

PPF was developed at the University of California, Santa Barbara, which utilizes the pluggable protocols framework to provide fault tolerance in CORBA [18]. It is an FT standard CORBA compliant infrastructure and achieves performance to maintain strong replica consistency, similar to DOORS or Eternal.

There is no need for any modification in CORBA ORB but PPF requires minimal modification in the application to run. It engages totem toolkit for totally ordered multicast of messages, fault detection and fault notification. FT protocol plug-in provides the fault tolerance on the server-side and client-side failover mechanisms. The Fault Detector, a component of FT protocol plug-in, detects the faults. Interoperability is achieved by passively replicated gateways, which provide access of un-replicated clients to replicated servers. Active and semi-active replication styles are supported for strong replica consistency. Smart duplicate mechanisms are provided for duplicate message detection and suppression. This scheme is similar to the interception approach as it employs an underlying toolkit for message delivery but in fact it is closer to the integration as fault tolerant mechanisms are embedded inside the ORB. But it differs from the integration-based systems as no need modification in ORB is required and it can be ported from one ORB to another.

5.12 CARRIAGE

CARRIAGE [19] is a fault tolerant CORBA system developed at the Southeast University of China, which employs portable interceptors to integrate *ORBUS* and *EDEN* to achieve fault tolerant services in CORBA. ORBUS is a CORBA implementation, and EDEN is a fault tolerant framework provided by IRISA/INRIA, France. Both of these were combined together on the basis of standardized Portable Interceptor mechanisms.

EDEN uses active replication style to enhance fault tolerance services. It consists Replication Manager, which handles all activities related to object replication, and Total Order Component, which is responsible for totally ordered multicast of messages. ORBUS, an OMG CORBA specification implementation that supports C++ and JAVA programming environments to work with distributed CORBA objects. ORBUS supports *ClientRequestInterceptor* for client-side and *ServerRequest- Interceptor* for server-side request processing. The approach followed is similar to the integration approach, as interceptors are hooked into the ORB; but differs from it as CARRIGE maintains inherent features of CORBA (i.e., language transparency, location transparency, portability and interoperability), which plays a vital role in its success. Moreover, it fully follows the standard specification and application programs do not require any modification to use this framework.

5.13 Lightweight Fault Tolerance (LW-FT)

Felber introduced a lightweight approach of embedding fault tolerance in existing CORBA system [20]. It employs replicated gateways for client-server interactions and uses semantic repository for achieving fault tolerance in CORBA. Use of gateways enables two fault tolerant CORBA frameworks to bridge that are supported by different mechanisms and QoS.

The proposed architecture uses client-side FT mechanisms and keeps semantic repository about server objects for fault tolerant request processing. The client request is propagated through replicated *Gateway*, which uses semantic repository for request processing. *Semantic repository* helps to choose optimal protocols for component interactions, replica management, automatic request redirection in case of failure, cache management to avoid unnecessary invocations to the servers, and load balancing of client requests. However, this approach cannot be applied to passively replicated or non-deterministic servers, and does not address the issues of maintaining strong replica consistency.

5.14 FRIENDS

FRIENDS stand for Flexible and Reusable Implementation Environment for your Next Dependable System [21]. FRIENDS, a meta-object protocol developed at LAAS-CNRS, Toulouse, provides libraries of meta-objects for fault tolerance, secure communication and group-based distributed applications. It exploits the reflective approach as the first time to build fault tolerance in CORBA systems. It aims to provide flexibility through object-oriented libraries of meta-objects and enhance non-functional requirements such as security by using the meta-objects.

FRIENDS system engages separate meta-objects for providing fault tolerance in CORBA. The system is composed of three layers, *Kernel layer*, *System layer* and *User layer*. System layer is responsible for providing fault tolerance by detecting crash faults, stable storage, secure communication, and replication management. User layer is responsible for controlling application objects and remote object interactions. System layer is built on the top of the Kernel layer, which is either a *UNIX* kernel or a micro kernel. Due to being kernel specific, it does not maintain portability. It uses time-outs to detect crash faults and both replication styles (active and passive) to maintain strong replica consistency. By applying FRIENDS, non-fault tolerant applications do not invoke functions on fault tolerant applications. The drawback of FRIENDS is that it is not CORBA compliant and fault tolerance properties cannot be configured dynamically as the link between objects and meta-objects cannot be changed at runtime.

5.15 FT-MOP

A Reflective fault tolerant CORBA system was developed at LAAS-CNRS, which uses a *Fault Tolerant Meta-Object Protocol* (FT-MOP) to build fault tolerance in CORBA [22]. By FT-MOP, desirable fault tolerance properties can be attached to CORBA objects as CORBA Meta Objects and enables off-the-shelf ORBs to be used. Its architecture is an extension of FRIENDS with the elimination of its drawbacks, which is based on a general-purpose runtime meta-object protocol. FT-MOP provides more efficient functionality by using a general-purpose compile-time MOP to implement a runtime MOP, than by using only a runtime MOP as in the FRIENDS system.

FT-MOP controls the behavior and the state of application level CORBA objects. FT-MOP handles the creation, deletion and invocation of CORBA objects. The client sends a request to the server by using the stub to invoke the server's services, which are implemented as IDL interfaces. The request is propagated to Metaobjects through the Metastub. Metaobjects controls the behavior and state of the server. FT-MOP is ORB compliant and it maintains interoperability of ORBs. FT-MOP is C++ language dependant, but the reuse ability of this system in many application domains with different object-oriented languages distinguishes it from other systems.

6 Comparative Analysis

Table 3 shows a comparative analysis of the existing FT-CORBA systems, which provides a quick insight on the features of these systems. Analysis parameters are: Approach, Interoperability, ORB compliance, OS dependence, Fault detection and notification, Replication transparency, Replication style, Replication implementation, Portability, Transparency to application, and FT-CORBA standard compliance. Values and meanings of these parameters are discussed above along with the systems.

Most of the FT-CORBA such as OGS, Eternal, DOORS, etc. provide fault monitoring based on non adaptive fault detectors [10], but their performance can be improved by using adaptive fault monitoring approaches, i.e., Discard Past Consider Present (DPCP) [23], or ADAPTATION [10] algorithms.

Table 3. Comparison among FT-CORBA Systems

Analysis Parameter	**DOORS**	**FTS**	**IRL**	**OGS**	**Newtop**	**AQuA**	**Aquarius**	**Eternal**	**Isis+Orbix**	**Electra**	**CARRIGE**	**PPF**	**Friends**	**FT-MOP**
Approach	service	service	service	service	service	service	service	interception	integration	integration	interception	integration	reflective	reflective
Interoperability	yes	yes	yes	yes	yes	no	yes	no	no	no	yes	yes	no	yes
ORB Compliance	yes	yes	yes	yes	yes	yes	yes	yes	no	no	yes	yes	yes	yes
OS dependence	no	no	no	no	no	yes	no	yes	yes	yes	no	no	yes	no
Fault detection and notification	combined	separate	separate	combined	combined	separate	separate	separate	combined	combined	separate	separate	separate	separate
Replication transparency	yes	yes	yes	no	no	yes	yes	yes	yes	yes	yes	yes	yes	Yes
Replication style	passive	both	passive	both	both	both	ordering protocol	both	both	passive	active	active, semi-active	metaobject protocol	metaobject protocol
Replication implementation	centralized	centralized	both	distributed	distributed	By Maestro /Ensemble	data-centric	by totem	by Isis	by Isis, Horus	EDEN	Totem	Open to programmer	open to programmer
Portability	yes	yes	yes	yes	yes	no	yes	no	no	no	yes	yes	no	yes
Transparency to application	not always	not always	yes	no	no	not always	yes	yes	yes	yes	yes	yes	yes	yes
FT-CORBA standard compliance	yes	No	no	no	no	no	no	yes	no	no	yes	yes	no	no

7 Conclusion

Traditional CORBA-based middleware cannot meet the demanding quality of service (QoS) for dependable systems, thus OMG fault tolerant CORBA specification addressed many of the QoS and fault tolerant mechanisms while maintaining CORBA's transparency, interoperability and simplicity of application programming. FT CORBA is not a replacement of fault tolerant infrastructure that were deployed before this specification, FT CORBA complements fault tolerant infrastructures by defining QoS policies, associated fault tolerance mechanisms and services to enhance the reliability of CORBA applications. This paper presents an overview of FT-CORBA specification; its architecture, requirements and limitations. We discussed the existing approaches for building CORBA based distributed systems, and evaluated the various fault tolerant CORBA systems by analyzing their prominent features and limitations. We have discussed the various styles of replicating the objects of the application that provides fault tolerance for CORBA applications.

Acknowledgement. The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4150/07E).

References

1. Felber, P., Narasimhan, P.: Experiences, Strategies, and Challenges in Building Fault-Tolerant CORBA Systems. IEEE Transactions on Computers 53(5) (May 2004)
2. Object Management Group: The Common Object Request Broker: Architecture and Specification, 2.6(edn.) OMG Technical Committee Document, formal/02-01-02 (January 2002)
3. Object Management Group: Fault Tolerant CORBA (Final Adopted Specification), OMG Technical Committee Document, formal/01-12-29 (December 2001)
4. Maffeis, S.: Run-Time Support for Object-Oriented Distributed Programming, PhD thesis, Univ. of Zurich (February 1995)
5. Felber, P.: The CORBA Object Group Service: A Service Approach to Object Groups in CORBA, PhD thesis, Swiss Federal Inst. of Technology, Lausanne (1998)
6. Natarajan, B., Gokhale, A., Yajnik, S., Schmidt, D.C.: Doors: Towards High-Performance Fault Tolerant CORBA. In: DOA 2000. Proceedings of the Second International Symposium on Distributed Objects and Applications, pp. 39–48 (February 2000)
7. Narasimhan, P.: Transparent Fault Tolerance for CORBA, PhD thesis, Dept. of Electrical and Computer Engineering, University of California, Santa Barbara (December 1999)
8. Narasimhan, P., Moser, L.E., Smith, P.M.: State Synchronization and Recovery for Strongly Consistent Replicated CORBA Objects. In: Proceedings of the 2001 International Conference on Dependable Systems and Networks, Goteborg, Sweden, pp. 261–270 (2001)
9. Marchetti, C., Mecella, M., Virgillito, A., Baldoni, R.: An Interoperable Replication Logic for CORBA Systems. In: DOA. Proceedings of the Second International Symposium on Distributed Objects and Applications, Belgium, pp. 21–23 (September 2000)
10. Sotoma, I.: ADAPTATION - Algorithms to ADAPTive fAulT monItOriNg and Their Implementation on CORBA. In: DOA. Proceedings of the Third International Symposium on Distributed Object and Applications, pp. 219–228. IEEE, Los Alamitos (2001)
11. IONA and Isis: An Introduction to Orbix+Isis, IONA Technologies Ltd. and Isis Distributed Systems, Inc. (1994)
12. Cukier, M., Ren, J., Sabnis, C., Sanders, W.H., Bakken, D.E., Berman, M.E., Karr, D.A., Schantz, R.: AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects. In: Proceedings of the 17th IEEE International Symposium on Reliable Distributed Systems, pp. 245–253 (October 1998)
13. Friedman, R., Hadad, E.: FTS: A high performance CORBA fault tolerance service. In: Proceedings of the IEEE Workshop on Object-Oriented Real-Time Dependable Systems, pp. 61–68. IEEE Computer Society Press, Los Alamitos (2002)
14. Marchetti, C., Virgillito, A., Baldoni, R.: Design of an Interoperable FT-CORBA Compliant Infrastructure. In: ERSADS 2001. Proceedings of the 4th European Research Seminar on Advances in Distributed Systems Dependable Systems, Bertinoro, Italy, pp. 14–18 (May 2001)

15. Morgan, G., Shrivastava, S., Ezhilchelvan, P., Little, M.: Design and Implementation of a CORBA Fault-Tolerant Object Group Service In:Proceedings of the Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (June 1999)
16. Chockler, G., Malkhi, D., Merimovich, B., Rabinowitz, D.: Aquarius: A Data-Centric approach to CORBA Fault-Tolerance. In: DOA. The workshop on Reliable and Secure Middleware, in the 2003 International Conference on Distributed Objects and Applications, Sicily, Italy (November 2003)
17. Chockler, G., Malkhi, D., Dolev, D.: Quorum Based Approach to CORBA Fault-Tolerance. In: ERSADS 2001. University Residential Center of University of Bologna, Bertinoro (Forlì), Italy, pp. 14–18 (May 2001)
18. Zhao, W., Moser, L.E., Melliar-Smith, P.M.: Design and implementation of a pluggable fault tolerant CORBA infrastructure. In: Proceedings of the International Parallel and Distributed Processing Symposium, Fort Lauderdale, pp. 35–44 (April 2002)
19. Goncalves, F., Greve, P., Hurfin, M., Narzul, J.-P.L.: OPEN EDEN: a Portable Fault Tolerant CORBA Architecture. In: Proceedings of the Second International Symposium on Parallel and Distributed Computing, p. 88. IEEE Computer Society Press, Los Alamitos (2003)
20. Felber, P.: Lightweight Fault Tolerance in CORBA. In: DOA 2001. Proceedings of the International Symposium on Distributed Objects and Applications, pp. 239–247 (September 2001)
21. Fabre, J.C., Pérennou, T.: A Metaobject Architecture for Fault Tolerant Distributed Systems: The FRIENDS Approach. IEEE Transactions on Computers, Special Issue on Dependability of Computing Systems 47(1), 78–95 (1998)
22. Killijian, M.O., Fabre, J.C., Ruiz-García, J.-C., Chiba, S.: A Metaobject Protocol for Fault-Tolerant CORBA Applications. In: 17th IEEE Symp. on Reliable Distributed Systems, West Lafayette, Indiana, USA, pp. 127–134 (1998)
23. Sotoma, I.: DPCP (Discard Past Consider Present) - A Novel Approach to Adaptive fault Detection in Distributed Systems. In: FTDCS 2001. Proceedings of the Eight IEEE Workshop on Future Trends of Distributed Computing Systems, IEEE C.S, Los Alamitos (2001)

Flexible Reuse of Middleware Infrastructures in Heterogeneous IT Environments

Ralf Wagner and Bernhard Mitschang

University of Stuttgart
{ralf.wagner, bernhard.mitschang}@ipvs.uni-stuttgart.de

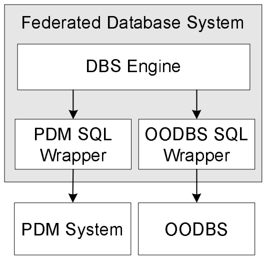
Abstract. Middleware systems and adapters integrate remote systems and provide uniform access to them. Middleware infrastructures consist of different types of middleware systems, e.g. application servers or federated database systems, and different types of adapters, e.g. J2EE connectors or SQL wrappers. Different adapter technologies are incompatible to each other, which requires to write new adapters where existing ones should be reused instead. Therefore, we introduce a virtualization tier that allows to uniformly handle and access adapters of different middleware platforms and that reuses existing adapter deployments, which avoids redundant administration tasks. Moreover, the virtualization tier can also reuse complete middleware infrastructures such that adapter deployments and adapter execution remains in the respective middleware system. This allows to flexibly reuse middleware infrastructures and facilitates the realization of new integration scenarios at reduced expense.

1 Introduction

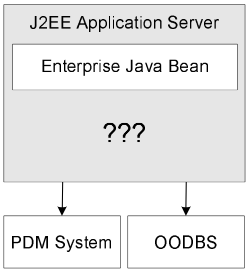
Middleware infrastructures are commonly used in heterogeneous IT environments. A middleware system can integrate diverse remote systems and offers applications uniform access to them. The integration part is realized by adapters that natively access remote systems. Adapters are plugged into a middleware system to enable the interoperation between middleware system and remote systems.

There are different middleware platforms and different adapter technologies. They range from research prototypes to commercial off-the-shelf (COTS) products or in-house integration platforms. Prominent examples of COTS are IBM WebSphere Message Broker, Microsoft Biztalk or SAP Netweaver. Such COTS middleware systems often support industry standards, e.g. the J2EE connector architecture [21], SQL Management of External Data (SQL/MED) [10] or the Web Services Architecture [3]. Well-known research prototypes are TSIMMIS [5], Information Manifold [12] or Garlic [20] to mention just a few of them.

Today, the tasks and processes of a company usually are tightly coupled with its IT infrastructure. These tasks and processes often change over time, which requires to reengineer the involved applications and IT infrastructure. Reengineering means to modify, extend or differently arrange and interconnect applications, middleware systems and other back-end systems. Such reengineering processes usually are complex and costly and therefore it would be beneficial to reuse the existing middleware infrastructure as far as possible instead of buying, installing and maintaining additional middleware infrastructure and re-implementing similar tasks over and over again.



(a) Federated DBS Using SQL Wrappers to Integrate Remote Systems.



(b) J2EE Application Server Demanding Access to Remote Systems.

Fig. 1. Integration Scenario

Therefore, we introduced a virtualization tier for reusing adapters of different middleware platforms [25]. However, this approach needs to additionally deploy adapters into the virtualization tier and it can only reuse adapters, but does not consider whole middleware systems. In this paper, we provide an approach that directly reuses adapters, which avoids additional deployments, and that allows to reuse complete middleware systems, which avoids the use of redundant middleware infrastructure.

In the next section we motivate why reuse of middleware infrastructures and especially of adapters is necessary and we give an example that shows how this can be achieved. In Section 3 we give a short overview of the virtualization tier and how it works in general. Section 4 analyzes different means of reusing adapters and middleware systems and evaluates the characteristics and applicability of these approaches. Section 5 discusses related work and Section 6 concludes the paper.

2 Motivation

The example scenario we discuss in this paper consists of two middleware systems used in different departments of a fictitious company. The first middleware system is shown in Figure 1(a). It is a federated database system (FDBS) that integrates a product data management (PDM) system and an OODBS by means of SQL wrappers. The second middleware system used by the company in a different application scenario is a J2EE application server that already integrates some other remote systems.

Business changes, which requires the second application scenario to be extended to adapt to these changes: the application server has to additionally integrate the PDM system and the OODBS. The question is how to access the remote systems (see Figure 1(b)).

2.1 Conventional Integration Solutions

The application server needs adapters to bridge the heterogeneities between application server and remote systems. J2EE application servers employ J2EE connectors to access remote systems, but they are not able to employ SQL wrappers of federated database systems. This is due to the differences in data model, processing model, programming model, APIs, etc. Thus, we cannot reuse the existing PDM SQL wrapper and OODBS

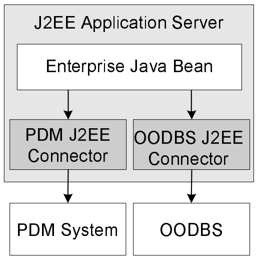


Fig. 2. Possible Integration Solution: Writing new J2EE Connectors

SQL wrapper, but we have to use suitable J2EE connectors, which are not available yet. A possible solution for this problem could be to write two new J2EE connectors, a PDM J2EE connector to integrate the PDM system and an OODBS J2EE connector to integrate the OODBS (see Figure 2).

However, in a previous paper we argued that writing a new adapter usually is an expensive, lengthy and error-prone task because the adapter programmer needs substantial knowledge about the involved middleware system and the integrated remote system, and especially about the adapter programming framework [25]. This knowledge comprises the data models, processing models, programming models, error models, quality of service requirements, etc. of the middleware platform and of the remote system as well as the adapter technology and the adapter programming framework.

In contrast, an existing adapter has already been written and can be used as is. An additional advantage is that the adapter has been tested and maintained in productive use and now works properly.

2.2 Reuse Issues

This means that we would like to reuse the PDM SQL wrapper and the OODBS SQL wrapper of Figure 1(a) instead of writing a new PDM J2EE connector and a new OODBS J2EE connector (cf. Figure 2). However, different adapter technologies usually are incompatible and thus the PDM SQL wrapper and the OODBS SQL wrapper do not work with the J2EE application server. We need some multiplexer functionality so that different middleware platforms can use different kinds of adapters. This is what we introduced with the virtualization tier (VT), which employs different kinds of adapters in the same way as the respective middleware platforms do and which additionally provides uniform access to these adapters [25].

3 Virtualization Tier

The solution of our integration problem in Figure 1(b) is shown in Figure 3: the existing PDM SQL wrapper and OODBS SQL wrapper are deployed into the VT and now are ready to execute requests issued by other middleware systems. In our example, the application server uses the VT J2EE connector to access the virtualization tier and thereby gains access to the PDM SQL wrapper and the OODBS SQL wrapper, which in turn

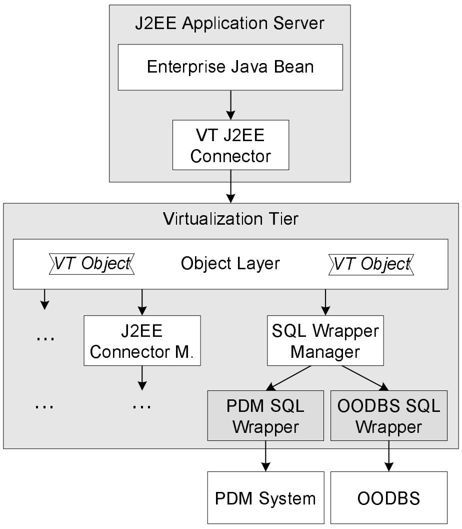


Fig. 3. Integration Solution Using the Virtualization Tier

access the PDM system and the OODBS, respectively. There is no need for writing a new PDM J2EE connector or a new OODBS J2EE connector.

3.1 VT Architecture

Figure 4 shows the architecture of the virtualization tier (VT). Adapter managers are responsible for handling and accessing adapters in the VT. For example, if a J2EE connector is deployed into the VT, the J2EE connector manager registers the J2EE connector in the VT and employs it during a VT request to access the corresponding remote

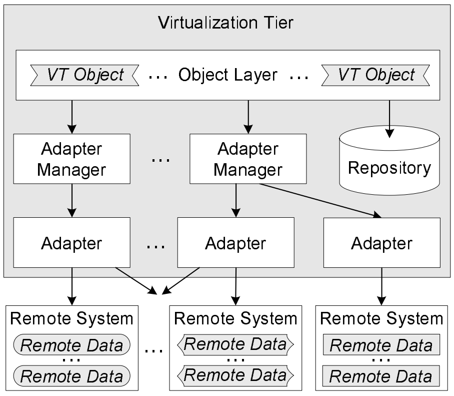


Fig. 4. Architecture of the Virtualization Tier

system. The SQL wrapper manager analogously registers and employs SQL wrappers and other adapter managers do the same for their adapters.

The object layer consists of VT objects that contain attributes and methods. They uniformly represent data and operations of remote systems. Client systems only need to access VT objects, which are resolved by the VT via the corresponding adapter managers, adapters and finally remote systems.

The object model of the VT has to support the needs of different adapter technologies. This means that it has to be able to represent data and operations and it has to be able to execute operations and access single data items as well as whole data sets, preferably in a set-oriented, declarative way. Therefore, we used the ODMG object model [4] to realize the VT object model and we also used the associated set-oriented, declarative object query language (OQL) to realize the VT access language. Hence, remote operations can be represented as VT object methods, and set-oriented queries in remote systems, e.g. SQL queries, can be represented as OQL queries in the VT.

3.2 Object Configurations

A VT object is defined by means of an object configuration, which consists of four configuration chapters. The chapters define the VT object and information about the adapter and the remote system. The VT stores the configuration chapters in its repository and uses them during runtime to resolve requested VT objects from remote systems. An object configuration comprises the following configuration chapters:

– The *Adapter Information Chapter* represents information about an adapter. An adapter manager uses an adapter information chapter to deploy an adapter and to access it during runtime.
  For example, a VT administrator deploys the PDM SQL wrapper into the VT by specifying the file path of the wrapper library, the file path of the PDM system client API library, and some other adapter configuration parameters.
– The *System Information Chapter* represents any information about a remote system. An adapter needs a system information chapter to access the remote system.
  For example, a VT administrator specifies the IP address and the port number of the PDM system daemon, the version of the PDM system, and some other remote system information.
– The *Object Information Chapter* correlates a VT object with a remote entity, e.g. a data structure or a procedure in a remote system.
  For example, a VT administrator specifies metadata about screw data structures in the PDM system: the struct name is *screw*, the fields are *len*, *dia*, *man*, *mat*, etc., the types of the attributes, the methods of the screw class, some access-related information, and how the screw class and its attributes and methods are mapped to VT object *VTScrew*.
– The *Object Definition Chapter* defines the attributes and methods of a VT object according to the ODMG object definition language (ODL) [4].
  For example, a VT administrator defines a VT object that is correlated to the screw object information chapter: VT object *VTScrew* with attributes *length*, *diameter*,

manufacturer, *material*, etc. and with some methods. The correlation between the PDM screw class and VT object *VTScrew* is completely specified in the object information chapter.

3.3 VT Processing

The VT client API allows to issue OQL requests to the VT as well as to directly access single VT objects. In both cases the VT retrieves the object configuration of the requested VT objects from the repository and determines the adapter managers that are responsible for resolving the VT objects. The VT tells the adapter managers which VT objects they have to resolve so that they can load the necessary adapters. Each adapter manager then issues adapter technology-specific requests to its adapters and the adapters in turn access the data and execute the operations in the remote systems.

The adapter technology-specific requests can be quite different. For example, the SQL wrapper manager transforms an OQL query into an SQL query for an SQL wrapper, the J2EE connector manager transforms an OQL query into a connector interaction and suitable input parameters and output parameters and executes the interaction on a J2EE connector, and a message broker manager transforms an OQL request into a business object request and issues it to a message broker adapter.

The example scenario in Figure 3 now works in the following way: if the Enterprise Java Bean (EJB) wants to access some screw objects in the PDM system, it has to access VT object *VTScrew* in the VT, which is representing the screw class in the PDM system. The EJB issues an OQL request to the VT via the VT J2EE connector. The VT determines that *VTScrew* is accessed and issues a corresponding request to the SQL wrapper manager, which in turn uses the PDM SQL wrapper to retrieve screw objects from the PDM system. The screw objects are transformed by the SQL wrapper manager into *VTScrew* object instances and finally returned to the application server. More details about the architecture of the VT and the execution of OQL requests can be found in [25].

4 Reuse of Adapters and Middleware Systems

The disadvantage of the solution shown in Figure 3 is that both adapters have to be deployed into the VT (*adapter information chapters*), that both remote systems have to be configured for proper access in the VT (*system information chapters*), and that the data and the operations in the remote systems that have to be accessed must be specified in the VT (*object information chapters*). This is additional work that should be avoided because the same deployments and definitions have already been provided for the SQL wrappers in the FDBS scenario in Figure 1(a). Moreover, if the same information is provided in the VT again, it is available in two places and also has to be maintained in two places.

Another point is that the VT must provide the same adapter deployment functionality and the same adapter execution functionality that the respective middleware system already contains. Providing the same functionality twice incurs higher costs, requires more computing power and leads to additional administration tasks and maintenance overhead. Therefore, we extend the VT approach to directly reuse existing middleware systems, adapters, deployments of adapters and definitions of remote systems, remote data and remote operations. Basically, there are two different kinds of direct reuse. The first one is reuse of adapters and adapter deployments (see Section 4.1) and the second one is reuse of middleware systems (see Section 4.2).

4.1 Reuse of Adapters and Adapter Deployments

Object configurations are used for properly resolving VT objects from remote systems by means of adapters. A VT administrator is responsible for creating object configurations. This task could be performed by corresponding middleware administrators as deployment tasks in a middleware system resemble deployment tasks in the VT.

Adapter Deployments. For example, an FDBS administrator deploys the SQL wrappers in the federated DBS scenario shown in Figure 1(a) and defines other information about the remote systems and their data and operations. The FDBS administrator is most suitable to act as an SQL wrapper deployer in the VT scenario where he has to specify the same information as in the federated DBS scenario, i.e. he deploys the SQL wrappers into the VT (*adapter information chapters*) and defines information about the remote systems (*system information chapters*) and the data and operations correlated with VT objects (*object information chapters*). The VT object definitions (*object definition chapters*) can be automatically derived from the object information chapters and further customized by the administrator if desired.

The information that is used for an adapter deployment in the VT, i.e. an object configuration, is the same as the information that is used for an adapter deployment in the respective middleware system. For example, the information that is used to define the configuration chapters for deploying the PDM SQL wrapper and the OODBS SQL wrapper into the VT (cf. Figure 3) is the same as the information that is used in the federated DBS scenario (cf. Figure 1(a)) to deploy the two wrappers into the FDBS. Hence, an adapter deployment in the VT depends on an adapter deployment in the respective middleware system, i.e. the same information is separately represented twice. But the best solution would be to have only one adapter deployment and the other one automatically generated based on the first one.

Automatic Extraction & Transformation. Let us come back to the scenario in Figure 1(b) where the application server has to access the PDM system and the OODBS. There are only SQL wrappers, the necessary J2EE connectors are not available. The originally proposed solution shown in Figure 3 requires that the federated DBS administrator deploys the PDM SQL wrapper and the OODBS SQL wrapper a second time, i.e. into the VT (cf. first adapter deployments in 1(a)). However, we do not want to do the same work a second time, we just want to take the deployments from the FDBS and transfer them in a suitable manner to the VT such that all necessary configuration chapters are automatically created.

The general process of transferring adapter deployments is shown in Figure 5 where the deployment information is extracted from the FDBS, transformed into configuration chapters, i.e. adapter information chapters, system information chapters, object information chapters, and one automatically generated object definition chapter for each

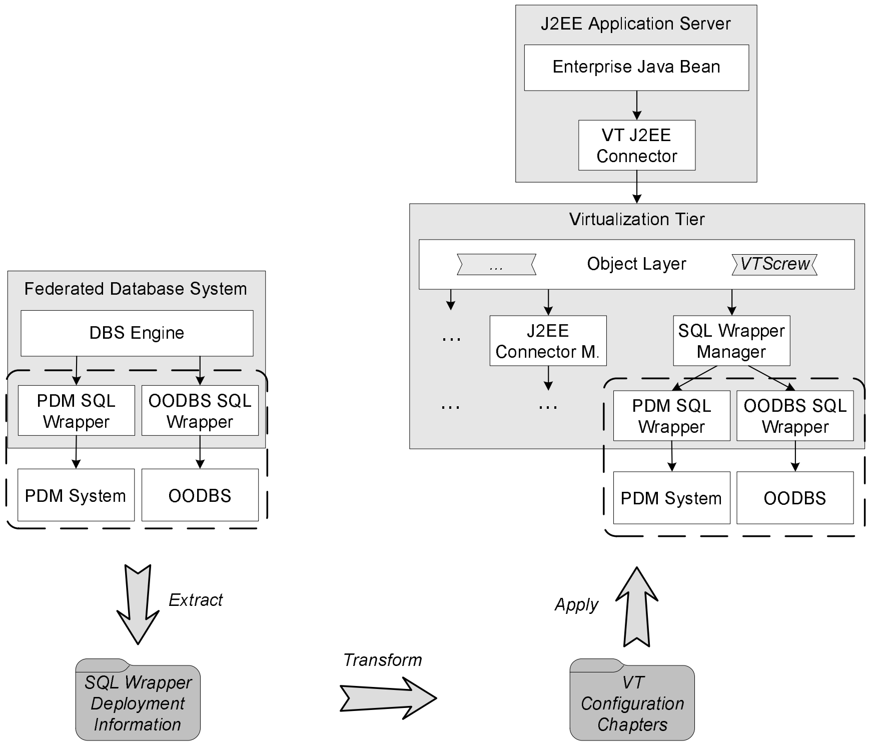
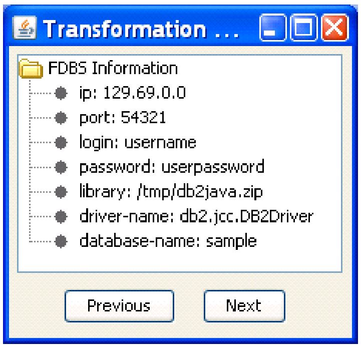


Fig. 5. Reusing Adapter Deployments

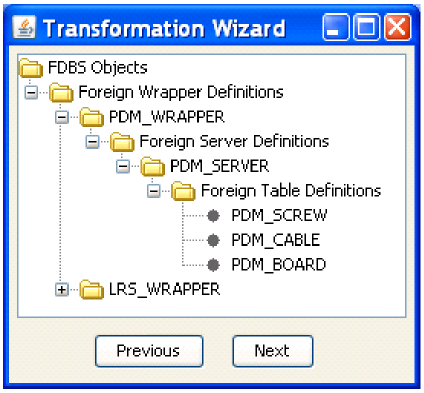
object information chapter representing data and operations in the PDM system or in the OODBS. Finally, the created configuration chapters are applied to the VT resulting in proper adapter deployments of the PDM SQL wrapper and of the OODBS SQL wrapper

The dashed box on the left side of Figure 5 represents the original deployment information of the SQL wrappers, remote systems and SQL tables. The dashed box on the right side represents the configuration chapters in the VT, which are derived from the original deployment information on the left side. This basically allows the application server to access the same entities in the PDM system and in the OODBS as the FDBS does when the application server uses the VT and the SQL wrappers that are deployed in the VT.

Deployment Transformation Wizards. The deployment transformation process is realized by means of *deployment transformation wizards*. An adapter manager provides a generic plug-in API so that deployment transformation wizards can be associated with the adapter manager. The VT repository stores all registered wizards and a VT administrator can choose among them to start a transformation process. In our example, an FDBS deployment transformation wizard is associated with the SQL wrapper manager. The transformation process here works as follows:



FDBS Information
- ip: 129.69.0.0
- port: 54321
- login: username
- password: userpassword
- library: /tmp/db2java.zip
- driver-name: db2.jcc.DB2Driver
- database-name: sample

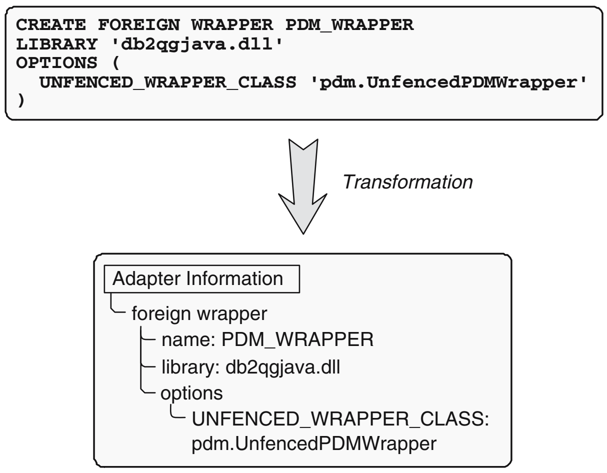


FDBS Objects
- Foreign Wrapper Definitions
  - PDM_WRAPPER
    - Foreign Server Definitions
      - PDM_SERVER
        - Foreign Table Definitions
          - PDM_SCREW
          - PDM_CABLE
          - PDM_BOARD
  - LRS_WRAPPER

Fig. 6. Example Dialog Windows of the FDBS Deployment Transformation Wizard

– A VT administrator starts the wizard, enters the IP address and the port number of the FDBS, a login name and a password, and some other information which are necessary for accessing the FDBS (see left wizard dialog window in Figure 6).
– The wizard then connects to the FDBS, looks up the FDBS catalog and displays the SQL wrappers deployed in the FDBS as well as the registered remote systems and the tables that represent data and operations of the remote systems (see right wizard dialog window in Figure 6).
– The VT administrator selects the entries of the PDM SQL wrapper and of the OODBS SQL wrapper, the entries of the PDM system and of the OODBS as well as the entries of some tables that represent data and operations in the PDM system and in the OODBS.
– The wizard retrieves the deployment information of the two SQL wrappers, the deployment information of the two remote systems and the deployment information of the selected tables from the FDBS catalog.
– The wizard transforms the retrieved FDBS catalog data into suitable configuration chapters (see Section “Example Transformation Process” below).
– The wizard transfers the wrapper libraries of the two SQL wrappers and other resources that are required for proper SQL wrapper execution, e.g. remote system API libraries, to the VT host.
– The wizard deploys the configuration chapters into the VT. This deploys the two SQL wrappers into the VT, registers the two remote systems in the VT and defines VT objects analogous to the tables in the FDBS.

Example Transformation Process. Figures 7 to 9 give an example of the transformation process. Several FDBS objects (represented as SQL statements) are transformed into corresponding VT information chapters. The definition of an SQL wrapper is shown in the upper part of Figure 7. There is a foreign wrapper named *PDM_WRAPPER* that uses a library called *db2qgjava.dll* and the hook class *pdm.UnfencedPDMWrapper* specified via a statement option. The adapter information chapter in the lower part of Figure 7 is automatically derived by the FDBS deployment transformation wizard from



```
CREATE FOREIGN WRAPPER PDM_WRAPPER
LIBRARY 'db2qgjava.dll'
OPTIONS (
  UNFENCED_WRAPPER_CLASS 'pdm.UnfencedPDMWrapper'
)
```

```
Adapter Information
└ foreign wrapper
    ├ name: PDM_WRAPPER
    ├ library: db2qgjava.dll
    └ options
        └ UNFENCED_WRAPPER_CLASS:
          pdm.UnfencedPDMWrapper
```

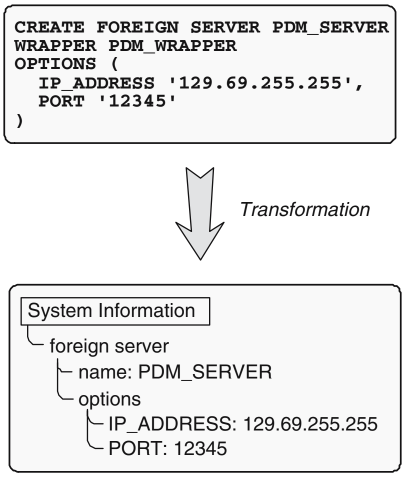
Fig. 7. SQL Create Statement of SQL Wrapper and Transformed SQL Wrapper Information Chapter

the foreign wrapper SQL statement. The adapter information chapter contains the same information as it is provided by the SQL statement.

The SQL statement in the upper part of Figure 8 defines a foreign server for the foreign wrapper of Figure 7. The foreign server is named *PDM_SERVER* and has two further properties, *IP_ADDRESS* and *PORT*, that tell the FDBS to which host it must connect to access the specified remote system. The FDBS deployment transformation wizard automatically transforms the foreign server SQL statement into the system information chapter shown in the lower part of Figure 8. The system information chapter contains the same information as the foreign server SQL statement provides.

Finally, the SQL statement in the upper part of Figure 9 creates a foreign table named *PDM_SCREW*, which represents screws information in the PDM system. The foreign table options in the second part of the SQL statement determine the API operation of the PDM system (*OPERATION*) and the parameters (*PARAMS*) that must be used to retrieve the screws information. The *TYPE* option specifies the data structure that holds the returned screws information and the column options of the foreign table columns specify the fields of the data structure that are correlated with the columns. The FDBS deployment transformation wizard automatically transforms the foreign table SQL statement into the object information chapter shown in the lower part of Figure 9, which contains all the information about the foreign table.

Usage of Adapter Deployments. If an SQL query requests foreign table *PDM_SCREW*, the FDBS accesses the PDM system using the wrapper functionality specified by the foreign wrapper SQL statement in Figure 7 and using the information about the PDM system specified by the foreign server SQL statement in Figure 8. Then the FDBS retrieves the data structure specified by the foreign table SQL statement of *PDM_Screw* in Figure 9.



```
CREATE FOREIGN SERVER PDM_SERVER
WRAPPER PDM_WRAPPER
OPTIONS (
  IP_ADDRESS '129.69.255.255',
  PORT '12345'
)
```

```
System Information
└ foreign server
   ├ name: PDM_SERVER
   └ options
      ├ IP_ADDRESS: 129.69.255.255
      └ PORT: 12345
```

Fig. 8. SQL Create Statement of Foreign Server and Transformed System Information Chapter

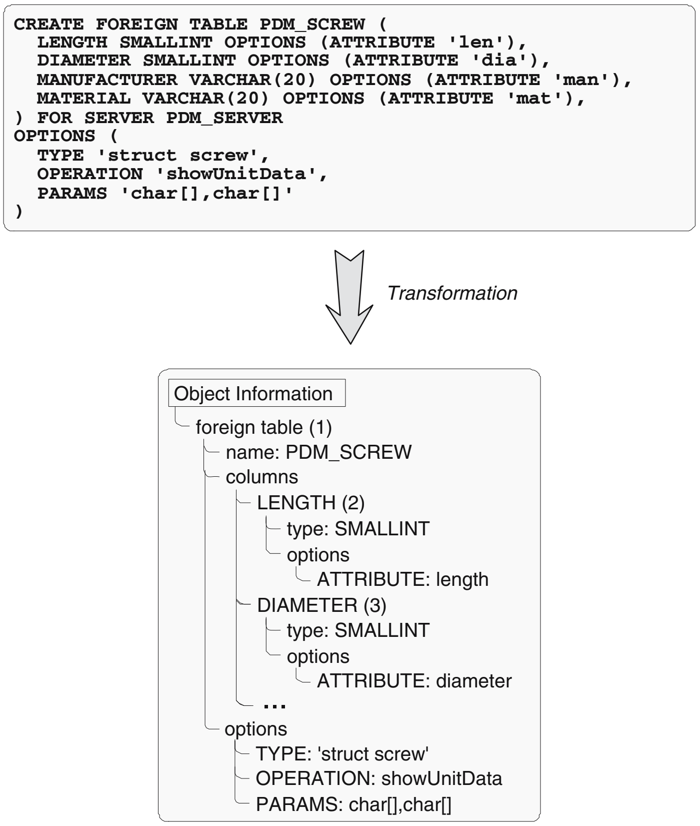
The VT works analogously: it accesses the PDM system using the wrapper functionality specified by the adapter information chapter in Figure 7 and using the information about the PDM system specified by the system information chapter in Figure 8. Then the VT retrieves the data structure specified by the object information chapter of *PDM_Screw* in Figure 9.

The result of this transformation process is that redundant adapter deployments are automatically generated and that they do not need to be manually created any longer. A VT administrator only uses the corresponding deployment transformation wizard to derive VT object definitions for the desired remote data and remote operations. In contrast to the automatic transformation process, the solution in Figure 3 requires new adapter deployments even if corresponding deployments already exist in a middleware system.

4.2 Reuse of Middleware Systems

If an adapter is automatically deployed into the VT based on an automatic adapter deployment transformation (cf. Section 4.1), still two adapter instances of the same adapter are available in the IT infrastructure, one adapter instance in the original middleware and one adapter instance in the VT. That is, the VT must incorporate functionality for executing the adapter in the VT, which is the same functionality that is required for executing the adapter in the middleware system. Put in other words: we use redundant middleware infrastructure, one in the original middleware system and one in the VT.

Redundant Middleware Infrastructure. For example, the scenario in Figure 5 allows to access a PDM SQL wrapper instance via the FDBS (shown on the left), but it also allows to execute the other PDM SQL wrapper instance via the VT (shown on the right). The same holds for the OODBS SQL wrapper. This kind of redundant middleware



```
CREATE FOREIGN TABLE PDM_SCREW (
  LENGTH SMALLINT OPTIONS (ATTRIBUTE 'len'),
  DIAMETER SMALLINT OPTIONS (ATTRIBUTE 'dia'),
  MANUFACTURER VARCHAR(20) OPTIONS (ATTRIBUTE 'man'),
  MATERIAL VARCHAR(20) OPTIONS (ATTRIBUTE 'mat'),
) FOR SERVER PDM_SERVER
OPTIONS (
  TYPE 'struct screw',
  OPERATION 'showUnitData',
  PARAMS 'char[],char[]'
)
```

```
Object Information
└ foreign table (1)
   ├ name: PDM_SCREW
   ├ columns
   │  ├ LENGTH (2)
   │  │  ├ type: SMALLINT
   │  │  └ options
   │  │     └ ATTRIBUTE: length
   │  ├ DIAMETER (3)
   │  │  ├ type: SMALLINT
   │  │  └ options
   │  │     └ ATTRIBUTE: diameter
   │  └ ...
   └ options
      ├ TYPE: 'struct screw'
      ├ OPERATION: showUnitData
      └ PARAMS: char[],char[]
```

Fig. 9. SQL Create Statement of Foreign Table and Transformed Object Information Chapter

infrastructure is not always intended from a global viewpoint. Intended redundant middleware infrastructure is used for purposes such as high availability, replication or distribution of computation, which is not the case here. The usage of redundant middleware infrastructure in cases like the given integration scenario leads to some disadvantages:

– higher software costs since functionality is implemented twice,
– higher hardware costs since more hardware is needed for executing the additional software,
– higher maintenance costs since this software and hardware must also be maintained.

Hence, it would be beneficial to reuse the existing middleware infrastructure for executing adapters. For example, we would like to use the FDBS for executing the PDM SQL wrapper and the OODBS SQL wrapper. In Figure 10, the VT uses the FDBS manager to access the FDBS and to execute the PDM SQL wrapper and the OODBS SQL wrapper indirectly since they are deployed in the FDBS, but no longer in the VT. In contrast, Figure 5 shows the former solution where the VT uses the SQL wrapper manager to execute the PDM SQL wrapper and the OODBS SQL wrapper directly, i.e. in the VT.

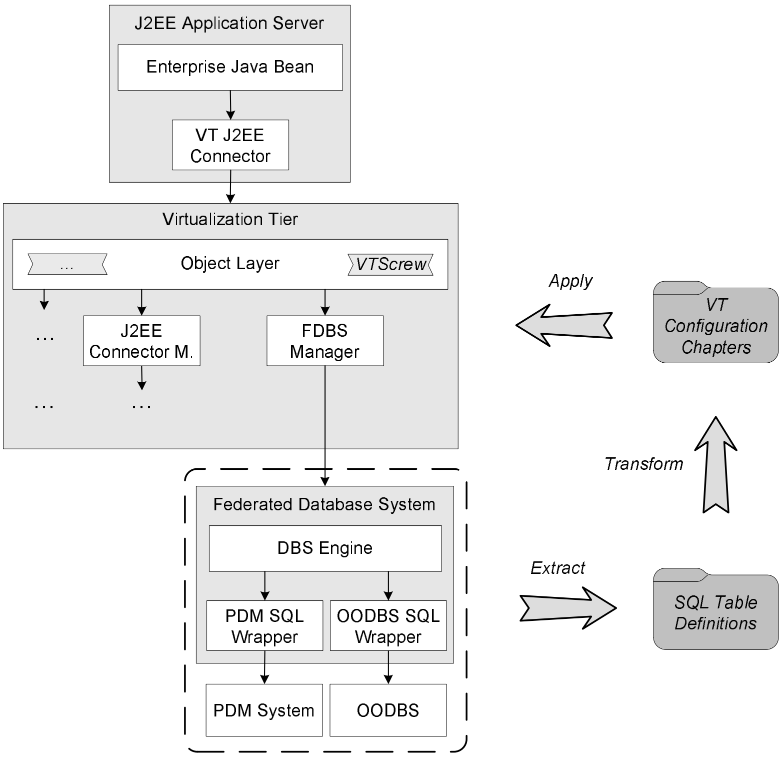
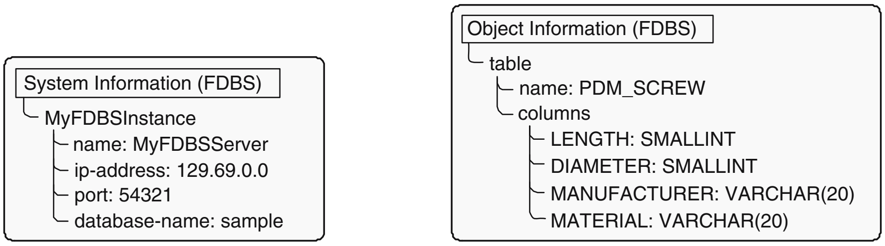


Fig. 10. Reusing Middleware Infrastructure

Different Adapter Deployments. This leads to differences in adapter deployments in both scenarios: the FDBS manager uses different configuration chapters than the SQL wrapper manager does. The FDBS manager does not need to know about how SQL wrappers are executed because the FDBS is executing the SQL wrappers. Thus, the adapter information chapter is no longer needed and can be completely omitted.



```
System Information (FDBS)
└ MyFDBSInstance
    ├ name: MyFDBSServer
    ├ ip-address: 129.69.0.0
    ├ port: 54321
    └ database-name: sample
```

(a) System Information Chapter of FDBS.

```
Object Information (FDBS)
└ table
    ├ name: PDM_SCREW
    └ columns
        ├ LENGTH: SMALLINT
        ├ DIAMETER: SMALLINT
        ├ MANUFACTURER: VARCHAR(20)
        └ MATERIAL: VARCHAR(20)
```

(b) Adapted Object Information Chapter for foreign table *PDM_SCREW*.

Fig. 11. FDBS Information Chapters

The PDM system and the OODBS as well as the data and the operations of the PDM system and of the OODBS are no longer visible to the VT, but only to the FDBS, which is responsible for properly accessing them via the SQL wrappers. Therefore, the system information chapters and object information chapters of the scenario in Figure 5 are not applicable here. Instead, the FDBS is the only remote system directly accessed by the VT so that we need only the FDBS system information chapter shown in Figure 11(a).

Thus, the VT only deals with the SQL-related part of the table definitions in the FDBS, but does not have to consider remote system-specific information specified in the options part of the foreign table SQL statements (cf. adapted object information chapter for foreign table *PDM_SCREW* in Figure 11(b)).

4.3 Homogenizing Middleware Infrastructure

Now we can reuse adapter deployment information in the VT and we can even reuse whole middleware systems so that we have only one adapter execution infrastructure and not any longer redundant infrastructure parts. An adapter is only deployed into the corresponding middleware system, but not into the VT. The VT only holds the information that is necessary for accessing the suitable middleware system and its data and operations.

Let us extend our integration scenario a bit further. In the next step the application server also has to integrate a supply chain management (SCM) system and a customer relationship management (CRM) system that are already integrated into a message broker (MB) by means of the SCM MB adapter and the CRM MB adapter, respectively (see dashed box in Figure 12). If we apply the middleware infrastructure reuse pattern of Section 4.2, we would come up with a message broker manager in the VT analogous to the FDBS manager. The message broker manager accesses the message broker and translates VT requests into message broker requests.

Middleware Adapters. However, the FDBS manager and the message broker manager contain identical functionality. For example, both managers have to access and deal with deployment information from the VT, they have to provide the plug-in mechanism for deployment transformation wizards, they need connection management functionality to handle server connections, etc.

Consequently, we put the functionality that is common to managers that are accessing middleware systems into a separate component, the *middleware adapter manager* (see Figure 12). The middleware adapter manager drives middleware adapters that contain the functionality required to access a specific middleware system and that translates VT requests into middleware-specific requests. For example, Figure 12 shows the FDBS middleware adapter and the message broker middleware adapter, which comprise the functionality that is left from the aforementioned FDBS manager and message broker manager, i.e. the parts that are required to properly access the middleware systems.

Deployment transformation wizards are then plugged into the middleware adapter manager and no longer directly into a middleware adapter, e.g. the FDBS deployment transformation wizard and the message broker deployment transformation wizard are plugged into the middleware adapter manager. The message broker deployment transformation wizard works analogous to the FDBS deployment transformation wizard: it

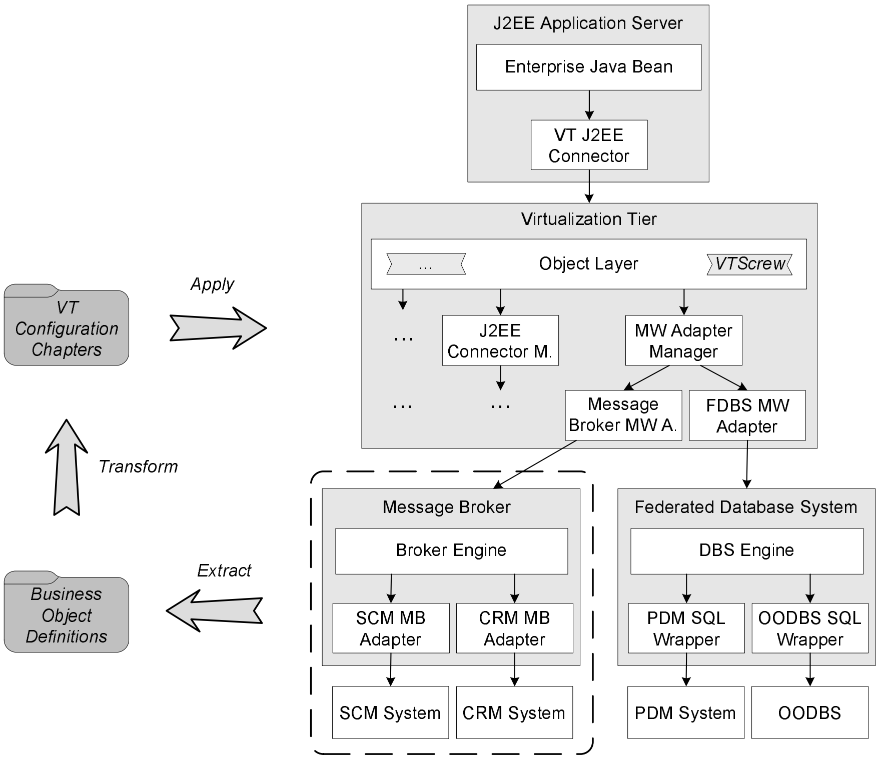


Fig. 12. Uniformly Handling Middleware Infrastructure

extracts the business object definitions from the message broker, transforms them into corresponding object information chapters and deploys them into the VT. Object definition chapters are automatically derived from the object information chapters and can be further customized by the VT administrator.

Middleware Adapter Deployment. Each middleware adapter needs an adapter information chapter so that the middleware adapter manager can properly access the middleware adapters. The VT administrator creates an adapter information chapter for a middleware adapter and thereby deploys the middleware adapter into the VT. Figure 13 shows the adapter information chapter for our FDBS.

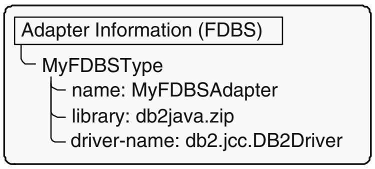

```
Adapter Information (FDBS)
└ MyFDBSType
   ├ name: MyFDBSAdapter
   ├ library: db2java.zip
   └ driver-name: db2.jcc.DB2Driver
```

Fig. 13. Adapter Information Chapter for the FDBS

Finally, the application server can access the four remote systems via the VT (cf. Figure 12). The SQL wrappers and the MB adapters remain in their respective middleware system and any required deployment information about them is extracted by deployment transformation wizards. The middleware infrastructure, i.e. the FDBS, the message broker and the deployed adapters, is reused as much as possible so that redundant middleware infrastructure and adapter deployments are completely avoided.

5 Related Work

Software reuse aims at using existing software artifacts during the software development process [24,7]. Software reuse is a wide field ranging from reuse of assembly language patterns to reuse of software system design structures [11].

Adapters are software components that can be reused in middleware systems of the same platform, e.g. J2EE connectors in application servers of different vendors. There have been many achievements in reusing software components in middleware systems, e.g. CORBA [15], CORBA Components [16], J2EE [23], Enterprise Java Beans [22], .NET/CLI [18,6], COM/DCOM [19]. However, reuse of software components, especially of adapters, in diverse middleware system has not been investigated so far. Adapters are based on specific integration technologies. They are inherently restricted to their respective middleware platform and cannot be directly reused in another middleware platform. This is the reason why systematic reuse of adapters and middleware infrastructure has not been investigated so far. However, this paper provides an approach that shows how this can be achieved.

Additionally, our approach is more promising than the generation of adapters, which is an attempt to flexibly deal with adapters. Such approaches aim at providing a high-level specification of an adapter's functionality so that the desired adapter code is generated or existing adapter libraries are suitably parameterized, e.g. see [1,2,8,9,13,17]. Adapter frameworks and adapter generation approaches inherently can handle only that parts of an adapter that are common to all adapters or that are at least similar for a group of adapters. The heterogeneity of remote systems would require generation approaches to flexibly deal with different access paradigms, request processing styles, data structures, data models, programming languages, APIs, etc. However, this complex task cannot be solved just by parameterizing a library or by specifying a set of declarative, high-level rules to generate the necessary code. Adapter generation only works if the targeted remote systems are restricted to a specific type so that their characteristics and properties are known in advance and can be considered in common libraries, rule sets or high-level scripting languages.

The recent evolution of universal metadata-driven generation approaches such as OMG's model-driven architecture (MDA) [14] could lead to techniques applicable to the generation of adapters, too. But there are no results for this kind of problem so far nor do we see any progress for that in the next time.

In contrast, the VT allows to systematically and dynamically reuse adapters and middleware infrastructures without affecting existing applications and without modifying existing middleware infrastructures.

6 Conclusion and Future Work

The use of different middleware platforms and different adapter technologies leads to incompatibilities and repeating programming efforts, i.e. writing adapters and usage of additional middleware infrastructure. Writing a new adapter is a costly task. Therefore, we proposed a virtualization approach for reusing existing adapters, adapter deployments and whole middleware infrastructures to reduce inconsistencies and additional costs.

In this paper, we showed how existing middleware infrastructure and adapter deployments can be directly reused in the VT. The VT approach allows to uniformly handle and access adapters of different middleware platforms without providing redundant deployments. Moreover, the VT approach can also reuse complete middleware infrastructures. This avoids the use of additional, redundant middleware infrastructure.

Important is that the VT can be smoothly used with existing IT infrastructures: their operation is not affected by the VT. Additionally, the VT approach provides for more flexibility in integration tasks. It leverages existing IT infrastructures the better the more middleware systems use the VT and the more adapters and middleware systems are reused by the VT.

Next, we want to automatically maintain correlated adapter deployments so that an adapter deployment in the VT that has been extracted from an adapter deployment in a middleware system is automatically changed if the adapter deployment in the middleware system is changed. The adapter deployment in the VT then needs not to be explicitly maintained by a VT administrator, but is automatically synchronized with the adapter deployment in the middleware system. We also want to automatically maintain the correlation between middleware systems and their reuse in the VT. If a middleware system configuration is changed, the corresponding configuration chapters in the VT have to be automatically adjusted to conform to the changes in the middleware system.

References

1. Ashish, N., Knoblock, C.A.: Semi-Automatic Wrapper Generation for Internet Information Sources. In: COOPIS 1997, pp. 160–169 (1997)
2. Baru, C., Gupta, A., Ludäscher, B., Marciano, R., Papakonstantinou, Y., Velikhov, P., Chu, V.: XML-Based Information Mediation with MIX. In: SIGMOD 1999, pp. 597–599 (1999)
3. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D. (eds.): Web Services Architecture. World Wide Web Consortium, W3C Working Group Note (February 2004)
4. Cattell, R.G.G., Barry, D.K., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T., Velez, F. (eds.): The Object Data Standard: ODMG 3.0. Morgan Kaufmann, San Francisco (2000)
5. Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J.D., Widom, J.: The TSIMMIS Project: Integration of Heterogeneous Information Sources. In: 16th Meeting of the Information Processing Society of Japan, pp. 7–18 (1994)
6. Ecma: Standard ECMA-335: Common Language Infrastructure (CLI), 6th edn. Ecma International (2006)
7. Freeman, P.: Software Reusability. IEEE, Los Alamitos (1987)

8. Gruser, J.-R., Raschid, L., Vidal, M.E., Bright, L.: Wrapper Generation for Web Accessible Data Sources. In: COOPIS 1998, pp. 14–23 (1998)
9. Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Breunig, M., Vassalos, V.: Template-Based Wrappers in the TSIMMIS System. In: SIGMOD 1997, pp. 532–535 (1997)
10. ISO. Information technology – Database languages – SQL – Part 9: Management of External Data (SQL/MED). International Organization for Standardization, 2nd edn., ISO/IEC 9075-9:2003 Published Standard (December 2003)
11. Krueger, C.W.: Software reuse. ACM Comput Surv. 24(2), 131–183 (1992)
12. Levy, A.Y.: The Information Manifold Approach to Data Integration. IEEE Intelligent Systems 13(5), 12–16 (1998)
13. Liu, L., Pu, C., Han, W.: XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In: ICDE 2000, pp. 611–621 (2000)
14. Miller, J., Mukerji, J. (eds.): MDA Guide Version 1.0.1. Object Management Group Inc (June 2003)
15. OMG. Common Object Request Broker Architecture: Core Specification. Object Management Group Inc. (March 2004)
16. OMG. CORBA Compnent Model Specification. Object Management Group Inc. (April 2006)
17. Raposo, J., Pan, A., Álvarez, M., Hidalgo, J., Viña, Á.: The Wargo System: Semi-Automatic Wrapper Generation in Presence of Complex Data Access Modes. In: Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds.) DEXA 2002. LNCS, vol. 2453, pp. 313–317. Springer, Heidelberg (2002)
18. Richter, J.: Applied Microsoft. In: NET Framework Programming, Microsoft Press, Washington (2002)
19. Rogerson, D.: Inside COM. Microsoft Press, Washington (1997)
20. Roth, M.T., Schwarz, P.M.: Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In: VLDB 1997, pp. 266–275 (1997)
21. Sun. *J2EE Connector Architecture Specification, Version 1.5.* Sun Microsystems Inc. (November 2003) (final release)
22. Sun. Enterprise JavaBeans, Version 3.0. Sun Microsystems Inc. (May 2006) (final release)
23. Sun. Java Platform, Enterprise Edition (Java EE) Specification, v5. Sun Microsystems Inc. (April 2006)(final release)
24. Tracz, W.: Software Reuse: Emerging Technology. IEEE, Los Alamitos (1988)
25. Wagner, R., Mitschang, B.: A Virtualization Approach for Reusing Middleware Adapters. In: ICEIS (2007)

Self-optimization of Clustered Message-Oriented Middleware

Christophe Taton[1], Noël De Palma[1], Daniel Hagimont[3], Sara Bouchenak[2], and Jérémy Philippe[1]

[1] Institut National Polytechnique de Grenoble, Grenoble, France
[2] Université Grenoble I, Grenoble, France
[3] Institut National Polytechnique de Toulouse, Toulouse, France
{Christophe.Taton,Noel.Depalma,Sara.Bouchenak, Daniel.Hagimont,Jeremy.Philippe}@inrialpes.fr

Abstract. Today's entreprise-level applications are often built as an assembly of distributed components that provide the basic services required by the application logic. As the scale of these applications increases, coarse-grained components will need be decoupled and will use message-based communication, often helped by Message-Oriented Middleware or MOMs.

In the Java world, a standardized interface exists for MOMs: Java Messaging Service or JMS. And like other middleware, some JMS implementations use clustering techniques to provide some level of performance and fault-tolerance. One such implementation is JORAM, which is open-source and hosted by the ObjectWeb consortium.

In this paper, we describe performance modeling of various clustering configurations and validate our model with performance evaluation in a real-life cluster. In doing that, we observed that the resource-efficiency of the clustering methods can be very poor due to local instabilities and/or global load variations.

To solve these issues, we provide insight into how to build autonomic capabilities on top of the JORAM middleware. Specifically, we describe a methodology to (i) dynamically adapt the load distribution among the servers (load-balancing aspect) and (ii) dynamically adapt the replication level (provisioning aspect).

Keywords: MOM, JMS, Autonomic management, Self-optimization.

1 Introduction

With the emergence of the internet, multiple applications require to be integrated with each other. One common glue technology for distributed, loosely coupled, heterogeneous software systems is Message-Oriented Middleware (MOM). MOMs are based on messages as the single structure for communication, coordination and synchronization, thus allowing asynchronous execution of components. Reliable communication is guaranteed by message queueing techniques that can be configured independently from the programming of software components. The Java community has standardized an interface for messaging (JMS). The use of MOMs in the context of internet has evidenced a need for highly scalable and highly available MOM. This paper analyses the performance of a MOM and proposes a self-optimization algorithm to improve the performance of the MOM infrastructure. This mechanism is based on a queue clustering solution : a *clustered queue* is a set of queues each running on different servers and sharing clients.

We will show that in some cases this mechanism can effectively provide a linear speedup but in other cases this mechanism is completely inefficient. We analyse that the efficiency of this mechanism depends on the distribution of client connections to MOM queues. We describe a solution that will improve the efficiency of this mechanism by optimizing the distribution of client connections in the cluster queue. Furthermore, an important aspect of this clustering policy is the selection of the level of clustering, i.e. the number of queues in the clustered queue. A commonly used solution is to select a fixed number of queues in the clustered queue. However, this static solution has some drawbacks. Let N be the (fixed) number of replicas. If N is too large, resources are wasted; if N is too small, performance may be compromised. In any case, the choice is problematic if the expected load of a queue is difficult to predict. Human administrators can monitor the load of the queuing system using adequate tools. However if a queue is underloaded or overloaded, an administrator cannot react as quickly as required.

This paper targets the optimization of these clustering mechanisms. This optimization will take place in two parts: (i) the optimization of the clustered queue load-balancing and (ii) the dynamic provisioning of a queue in the clustered queue.

The first part allows the overall improvement of the clustered queue performance while the second part optimizes the resource usage inside the clustered queue. Thus the idea is to create an autonomic system that:

– fairly distributes client connections among the queues belonging to the clustered queue,
– dynamically adds and removes queues in the clustered queue depending on the load. This would allow us to use the adequate number of queues at any time.

This paper is organized as follow: Sections 2 and 3 present the context of this work. Section 4 details the different cases that may occur with a clustered queue. Sections 5 and 6 present the control rules and the control loop. Section 7 shows performance evaluation. Finally section 8 presents related work and section 9 draws a conclusion and outlines future work.

2 Background: Java Message Service (JMS)

JMS is part of Sun's J2EE platform. It provide a programming interface (API) to interconnect different applications through a messaging middleware. The JMS architecture identifies the following elements:

– **JMS provider:** an implementation of the JMS interface for a Message Oriented Middleware (MOM). Providers are implemented as either a Java JMS implementation or an adapter to a non-Java MOM.
– **JMS client:** a Java-based application or object that produces and/or consumes messages.

- **JMS producer:** a JMS client that creates and sends messages.
- **JMS consumer:** a JMS client that receives messages.
- **JMS message:** an object that contains the data being transferred between JMS clients.
- **JMS queue:** a staging area that contains messages that have been sent and are waiting to be read. As the name queue suggests, the messages are delivered in the order they are sent. A message is removed from the queue once it has been read.
- **JMS topic:** a distribution mechanism for publishing messages that are delivered to multiple subscribers.
- **JMS connection:** A connection represents a communication link between the application and the messaging server. Depending on the connection type, connections allow users to create sessions for sending and receiving messages from a queue or topic.
- **JMS session:** Represents a single-threaded context for sending and receiving messages. A session is single-threaded so that messages are serialized, meaning that messages are received one-by-one in the order sent.

For our experiments we chose JORAM (Java Open Reliable Asynchronous Messaging). It is open source software released under the LGPL license which incorporates a 100% pure Java implementation of JMS. JORAM adds interesting extra features to the JMS API such as the clustered queue mechanisms. The following section describes the mechanism of queue clustering.

3 Clustered Queues

The clustered queue feature provides a load balancing mechanism. A clustered queue is a cluster of queues (a given number of queue destinations knowing each other) that are able to exchange messages depending on their load.

Each queue of a cluster periodically reevaluates its load factor and sends the result to the other queues of the cluster. When a queue hosts more messages than it is authorized

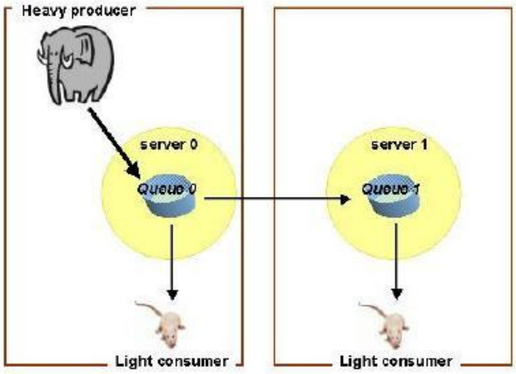


Fig. 1. A queue cluster

to do, and according to the load factors of the cluster, it distributes the extra messages to the other queues. When a queue is requested to deliver messages but is empty, it requests messages from the other queues of the cluster. This mechanism guarantees that no queue is hyper-active while some others are lazy, and tends to distribute the work load among the servers involved in the cluster. The figure above shows an example of a cluster made of two queues. An heavy producer accesses its local queue (queue 0) and sends messages. The queue is also accessed by a consumer but requesting few messages. It quickly becomes loaded and decides to forward messages to the other queue (queue 1) of its cluster, which is not under heavy load. Thus, the consumer on queue 1 also gets messages, and messages on queue 0 are consumed in a quicker way.

4 Clustered Queue Load-Balancing

We present in this section the key parameters that influence the behavior and the performance of a clustered queue. In the first part, we show the impact of the distribution of clients connections on the performance; in the second part, we provide some details about resource provisioning.

4.1 Configuration of Clients Connections

Standard queue. A standard single queue Q_i is connected to N_i message producers that induce a message production rate p_i, and to M_i message consumers that induce a message consumption rate c_i. The queue length l_i denotes the number of messages waiting to be read in the queue; l_i is always positive and obeys to the law :

$$\Delta l_i = p_i - c_i$$

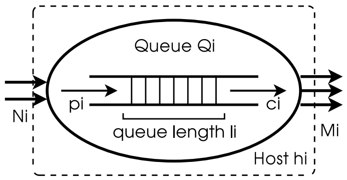


Fig. 2. Standard JMS queue Q_i

Depending on the ratio between message production and message consumption, three cases are possible:

- $\Delta l_i = 0$: message production and message consumption annihilate themselves and queue length l_i is constant. Queue Q_i is said to be *stable*.
- $\Delta l_i > 0$: there is more message production than message consumption. Queue Q_i will grow and eventually saturate as the queue length l_i gets too big. Queue Q_i is then *unstable* and is said to be *flooded*. Once the queue saturates, the message production rate of producers will be limited. The queue then stabilizes with $\Delta l_i = 0$.

– $\mathit{\Delta} l_i < 0$: there is more message consumption than message production in the queue. Queue length l_i decreases down to 0; the queue is *unstable* and said to be *draining*. Once queue Q_i is empty, message consumers will have to wait and become lazy, Q_i will stabilize with $\mathit{\Delta} l_i = 0$.

The message production and consumption rates are in direct relationships with the number of message producers and consumers:

$$p_i = f(N_i)$$
$$c_i = g(M_i)$$

Thus the stability of a standard single queue is controlled by the ratio between the number of message producers and the number of message consumers.

Clustered queue. Clustered queues are standard queues that share a common pool of message producers and consumers, and that can exchange message to balance the load. All the queues of a clustered queue are supposed to be directly connected to each other. This allows message exchanges between the queues of a cluster in order to empty flooded queues and to fill draining queues.

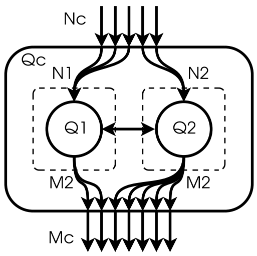


Fig. 3. Clustered queue Q_c

The clustered queue Q_c is connected to N_c message producers and to M_c message consumers. Q_c is composed of standard queues $Q_i (i \in [1..k])$. Each queue Q_i is in charge of a subset of N_i message producers and of a subset of M_i message consumers:

$$\begin{cases} N_c = \sum_i N_i \\ M_c = \sum_i M_i \end{cases}$$

The distribution of the clients between the queues Q_i is described as follows: x_i (resp. y_i) is the fraction of message producers (resp. consumers) that are directed to Q_i.

$$\begin{cases} N_i = x_i \cdot N_c \\ M_i = y_i \cdot M_c \end{cases}, \begin{cases} \sum_i x_i = 1 \\ \sum_i y_i = 1 \end{cases}$$

The standard queue Q_i to which a consumer or producer is directed to cannot be changed after the client connection to the clustered queue. This way, the only action that may affect the client distribution among the queues is the selection of an adequate queue when the client connection is opened.

The clustered queue Q_c is characterized by its aggregate message production rate p_c and its aggregate message consumption rate c_c. The clustered queue Q_c also has a virtual clustered queue length l_c that aggregates the length of all contained standard queues:

$$l_c = \sum_i l_i = p_c - c_c, \begin{cases} p_c = \sum_i p_i \\ c_c = \sum_i c_i \end{cases}$$

The clustered queue length l_c obeys to the same law as a standard queue:

– Q_c is globally stable when $\Delta l_c = 0$. This configuration ensures that the clustered queue is globally stable. However Q_c may observe local unstabilities if one of its queues is draining or is flooded.
– If $\Delta l_c > 0$, the clustered queue will grow and eventually saturate; then message producers will have to wait.
– If $\Delta l_c < 0$, the clustered queue will shrink until it is empty; then message consumers will also have to wait.

We now suppose that the clustered queue is globally stable, and we list various scenarios that illustrate the impact of client distribution on performance.

Optimal client distribution of the clustered queue Q_c is achieved when clients are fairly distributed among the k queues Q_i. Assuming that all queues and hosts have equivalent processing capabilities and that all producers (resp. consumers) have equivalent message production (resp. consumption) rates (and that all produced messages are equivalent : message cost is uniformly distributed), this means that:

$$\begin{cases} x_i = 1/k \\ y_i = 1/k \end{cases}, \begin{cases} N_i = \frac{N_c}{k}, \\ M_i = \frac{M_c}{k} \end{cases}$$

In these conditions, all queues Q_i are stable and the queue cluster is balanced. As a consequence, there are no internal queue-to-queue message exchanges, and performance is optimal. Queue clustering then provides a quasi-linear speedup.

The worst clients distribution appears when one queue only has message producers or only has message consumers. In the example depicted on Figure 3, this is realized when:

$$\begin{cases} x_1 = 1 \\ y_1 = 0 \end{cases}, \begin{cases} x_2 = 0 \\ y_2 = 1 \end{cases}, \begin{cases} N_1 = N_c \\ M_1 = 0 \end{cases}, \begin{cases} N_2 = 0 \\ M_2 = M_c \end{cases}$$

Indeed, this configuration implies that the whole message production is directed to queue Q_1. Q_1 then forwards all messages to Q_2 that in turn delivers messages to the message consumers.

Local instability is observed when some queues Q_i of Q_c are unbalanced. This is characterized by a mismatch between the fraction of producers and the fraction of consumers directed to Q_i:

$$x_i \neq y_i$$

In the example showed in Figure 3, Q_c is composed of two standard queues Q_1 and Q_2. A scenario of local instability can be envisioned with the following clients distribution:

$$\begin{cases} x_1 = 2/3 \\ y_1 = 1/3 \end{cases}, \begin{cases} x_2 = 1/3 \\ y_2 = 2/3 \end{cases}$$

This distribution implies that Q_1 is flooding and will have to enqueue messages, while Q_2 is draining and will see its consumer clients wait. However the queue cluster Q_c ensures the global stability of the system thanks to internal message exchanges from Q_1 to Q_2.

A stable and unfair distribution can be observed when the clustered queue is globally and locally stable, but the load is unfairly balanced within the queues. This happens when the client distribution is non-uniform.

In the example presented in Figure 3, this can be realized by directing more clients to Q_1 than Q_2:

$$\begin{cases} x_1 = 2/3 \\ y_1 = 2/3 \end{cases}, \begin{cases} x_2 = 1/3 \\ y_2 = 1/3 \end{cases}$$

In this scenario, queue Q_1 processes two third of the load, while queue Q_2 only processes one third. Suc situation can lead to bad performance since Q_1 may saturates while Q_2 is lazy.

It is worthwhile to indicate that these scenarios may all happen since clients join and leave the system in an uncontrolled way. Indeed, the global stability of a (clustered) queue is under responsability of the application developper. For instance, the queue can be flooded for a period; we then assume that it will get inverted and draining after, thus providing global stability over time.

4.2 Provisioning

The previous scenario of stable and non-optimal distribution raises the question of the capacity of a queue.

The capacity C_i of standard queue Q_i is expressed as an optimal number of clients. The queue load L_i is then expressed as the ratio between its current number of clients and its capacity:

$$L_i = \frac{N_i + M_i}{C_i}$$

– $L_i < 1$: queue Q_i is underloaded and thus lazy; the message throughput delivered by the queue can be improved and ressources are wasted.
– $L_i > 1$: queue Q_i is overloaded and may saturate; this induces a decreased message throughput and eventually leads to thrashing.
– $L_i = 1$: queue Q_i is fairly loaded and delivers its optimal message throughput.

These parameters and indicators are transposed to queue clusters. The clustered queue Q_c is characterized by its aggregated capacity C_c and its global load L_c:

$$C_c = \sum_i C_i \ , \ L_c = \frac{N_c + M_c}{C_c} = \frac{\sum_i L_i \cdot C_i}{\sum_i C_i}$$

The load of a clustered queue obeys to the same law as the load of a standard queue.

However a clustered queue allows us to control k, the number of inside standard queues, and thus to control its aggregated capacity $C_c = \sum_{i=1}^{k} C_i$. This control is indeed operated with a re-evaluation of the clustered queue provisioning.

- When $L_c < 1$, the clustered queue is underloaded: if the clients distribution is optimal, then all the standard queues inside the cluster will be underloaded; however, as the client distribution may be non-optimal, some of the single queues may be overloaded, even if the cluster is globally lazy. If the load is too low, then some queues may be removed from the cluster.
- When $L_c > 1$, the clustered queue is overloaded: even if the distribution of clients over the queues is optimal, there will exist at least one standard queue that will be overloaded. One way to handle this case is to re-provision the clustered queue by inserting one or more queues into the cluster.

5 A Self-optimizing Clustered Queue

In this section, we present the design of an autonomic ability which targets the optimization of a clustered queue. The optimization takes place in two steps : (i) the optimal load-balancing of a clustered queue, and (ii) the dynamic provisioning of queues in a clustered queue.

The first part allows the overall improvement of the clustered queue performance while the second part optimizes the queue resource usage inside the clustered queue. Thus the idea is then to create an autonomic system that :

- fairly distribute client connections to the pool of server hosts in the clustered queue,
- dynamically adds and removes queues in a clustered queue depending on the load. That would allow us to use the adequate number of queues at any time.

The implementation of these optimizations relies on the model of clustered queue performance which has been presented in the previous sections.

5.1 Control Rules

The global clients distribution D of the clustered queue Q_c is captured by the fractions of message producers x_i and consumers y_i. The optimal clients distribution D_{opt} is realized when all queues are stable ($\forall i \ x_i = y_i$) and when the load is fairly balanced over all queues ($\forall i, j \ x_i = x_j, \ y_i = y_j$). This implies that the optimal distribution is reached when $x_i = y_i = 1/k$.

$$D = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_k & y_k \end{bmatrix} \ , \ D_{opt} = \begin{bmatrix} 1/k & 1/k \\ \vdots & \vdots \\ 1/k & 1/k \end{bmatrix}$$

Local instabilities are characterized by a mismatch between the fraction of message producers x_i and consumers y_i on a standard queue. The purpose of this rule is the stability of all standard queues so as to minimize internal queue-to-queue message transfert.

(R_1) $x_i > y_i$: Q_i is flooding with more message production than consumption and should then seek more consumers and/or fewer producers.
(R_2) $x_i < y_i$: Q_i is draining with more message consumption than production and should then seek more producers and/or fewer consumers.

Load balancing rules control the load applied to a single standard queue. The goal is then to enforce a fair load balancing over all queues.

(R_3) $L_i > 1$: Q_i is overloaded and should avoid accepting new clients as it may degrade its performance.
(R_4) $L_i < 1$: Q_i is underloaded and should request more clients so as to optimize resource usage.

Global provisioning rules control the load applied to the whole clustered queue. These rules target the optimal size of the clustered queue while the load applied to the system evolves.

(R_5) $L_c > 1$: the queue cluster is overloaded and requires an increased capacity to handle all its clients in an optimal way.
(R_6) $L_c < 1$: the queue cluster is underloaded and could accept a decrease in capacity.

5.2 Algorithm

This section presents an algorithm for the self-optimization of queue clustering systems. As a first step we do not allow the modification of the underlying middleware. This constraint restricts the control mechanisms that we can use to implement the autonomic behaviour.

System events and controls. Without modification, the underlying JMS middleware does not provide facilities such as session migration that would allow us to migrate clients from one queue to another. However clustered queue systems allow the control of the queue that will handle a new message producer (resp. consumer). This control translated in the model terms means that some x_i (resp. y_i) will be increased, and we have the choice for i.

On the contrary, a message producer (resp. consumer) that leaves the system induces an unavoidable and uncontrolled decrease in some x_i (resp. y_i).

Thus a clustered queue system generates 4 types of events that we can use to control and optimize the system:

join(Producer)	join(Consumer)
leave(Producer, Q_i)	leave(Consumer, Q_i)

Algorithm 1. Client joining algorithm

```
on join(ClientType ∈ {Producer, Consumer}, $Q_c$)
if ($L_c \geq 1$) then
    // Queue cluster will be overloaded
    // An additional queue is required
    $Q_{k+1} \leftarrow$ NewQueue()
    AddQueue($Q_c$, $Q_{k+1}$)
end if
$Q_i =$ ElectQueue($Q_c$, ClientType)
return CreateSession(ClientType,$Q_i$)
```

Algorithm 2. Client leaving algorithm

```
on leave(ClientType ∈ {Producer, Consumer}, $Q_i \in Q_c$)
if (IsMarked($Q_i$, "to be removed") and IsEmpty($Q_i$) then
    RemoveQueue($Q_c$, $Q_i$)
    DestroyQueue($Q_i$)
end if
if ($L_c < 1$) then
    $Q_i =$ ElectRemovableQueue($Q_c$)
    if $Q_i \neq null$ then
        Mark($Q_i$, "to be removed")
    end if
end if
```

The control rules must then be implemented as handlers to these events. The algorithms that control the distribution of clients and the queue cluster provisioning are depicted in Algorithms 1 and 2.

The ElectQueue(ClientType) function chooses the queue that is most far away from the targeted client distribution. The elected queue Q_i then maximizes the gap to the optimal. When considering a new client that is a message producer (resp. consumer), the gap is evaluated with $1/k - x_i$ (resp. with $1/k - y_i$). Thus Q_i satisfies:

$$\begin{cases} x_i = \min_j x_j \text{ (when ClientType = Producer)} \\ y_i = \min_j y_j \text{ (when ClientType = Consumer)} \end{cases}$$

The ElectRemovableQueue(Q_c) chooses one queue that can be removed from the queue cluster. A queue cannot be removed on demand since it may still have clients connected to it: a queue can only be removed when its last client decides to leave. Thus the removal of a queue Q_i will need two steps: (1) Q_i is marked "to be removed" and no more clients will be addressed to it; (2) when Q_i's last client leaves, Q_i can then be removed from the cluster. Moreover, even if Q_c is underloaded, queue Q_i should not be removed if its removal let Q_c be overloaded. Thus the condition to allow Q_i's removal is:

$$C_i \leq C_c - (N_c + M_c)$$

The following section gives implementation details about these algorithms.

6 Implementation Details

6.1 Requirements

To implement a self-managed queue cluster using the autonomic computing design principles require the following management capabilities:

- to know the current number of message producers and consumers,
- to know where the servers are deployed, where the queues are deployed and what is their configuration,
- to route a new client connection to the best queue to reach the optimal,
- to detect the overload or the underload of a queue cluster,
- to allocate a new server to create a new queue,
- to add and remove a queue in a server.

6.2 The Control Loop

To simplify, we will consider that clients create only one session by connection. By doing this we assimilate the creation of sessions and the creation of connections. Assuming this, the first prototype is achieved by wrapping the standard JMS Connection-Factory by a ”LBConnectionFactory” (where LB stands for Load Balancing).

LBConnectionFactory. As the client gets the connection factory through JNDI, it gets the LBConnectionFactory instead. This is the main non-functionnal hook in the system that allows to control the distribution of producers and consumers among servers. This component offers the following methods:

createConnection(...) takes the type of the client as a parameter (Producer or Consumer). To create the connection with the right server, it requests a component called “ClusterManager” which provisions (“resizes”) the cluster and elects a server according to the current state of the system (the servers, the load of each queue in terms of producers and consumers).

closeConnection(...) effectively closes the connection to the server and notifies the ClusterManager so it can decrease the number of queues in the cluster if necessary.

ClusterManager. This component stores the state of the global system, i.e. the number of servers currently used, the number of clients connected to each server, their type. The state changes as client requests are received from the LBConnectionFactory. The different requests are:

- a consumer wants a connection;
- a producer wants a connection;
- a consumer wants to close a connection on server Q_i;
- a producer wants to close a connection on server Q_i.

In the first two cases, the ClusterManager elects a server taking into account the capacities in terms of clients. If the cluster is evaluated to be full of producers or consumers, the LBClusterManager uses the procedures **NewQueue()** and **AddQueue()** to launch a JORAM server on a free host and to create a queue linked to the cluster on that server. Of course, the cluster manager will update its internal image of the global system according to this.

7 Evaluation

A series of experiments was run to assess the performance of JORAM. Rather than finding an absolute maximum, these experiments were aimed at finding the relevant factors impacting the performance of JORAM queues. The focus was on assessing the usefulness of using queue clusters instead of single queues.

Environment. The experiments presented below were run on a cluster of Mac Mini computers with the following specifications:

- *Mac OS X 10.4.7, Intel Core Duo 1.66 GHz, 2 GB SDRAM DDR2 (667 MHz frontal bus)*
- *Java J2SDK1.4.2_13, JORAM 4.3.21*
- *Ethernet Gigabit network*

In each experiment, the measurements were taken with JMX probes located on a computer outside the cluster. Each JORAM queue ran a JMX server which was accessed by one of the JMX probes. The monitored attributes on the queue were *NbMsgsDeliverSinceCreation* which is the number of messages read by consumers on the queue since its creation and *MessageCounter* which is the number of messages presently waiting in the queue. The JMX probes were reading these attributes every second.

In the following experiments, each JORAM queue was located on a distinct node. The queues were running in a non persistent configuration. The producers and consumers were transactional with a commit between each message. The Java Virtual Machine hosting each queue was able to use 1536 MBytes of memory. The Garbage Collector was disabled to prevent random hits on performance. The size of the JMS messages used was 1 KBytes. The network was not considered to be meaningful factor in these experiments.

To obtain meaningful results, each experiment was run three times. The charts were constructed using the average of the three tests. The average throughput was calculated excluding the first five and last five seconds as a way to only account for the stable part of the process.

The number of waiting messages factor. This experiment shows the impact the number of messages waiting in the queue on the performance. Producers were writing messages in a single queue for a duration of 60 seconds then consumers were reading these messages from the queue until it was empty.

Figure 4 shows this experiment. We observe that the number of messages waiting in the queue has a strong impact on the performance: the message processing rate of the queue decreases as the queue length grows.

Single queue limit. In order to assess the interest of having a cluster queue instead of a single queue, we need to measure the highest throughput a single queue can reach with the previously described parameters. We made multiple measurements with a varying number of producers and consumers accessing a single queue. For a given number of producers, the ratio to obtain the best throughput was always 1 producer for 2 consumers. These measurements are summed up on Figure 5.

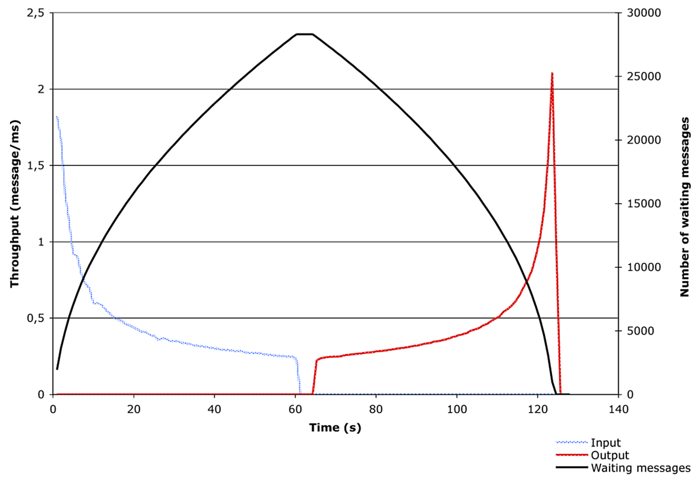


Fig. 4. Impact of the Waiting Messages on the Performance

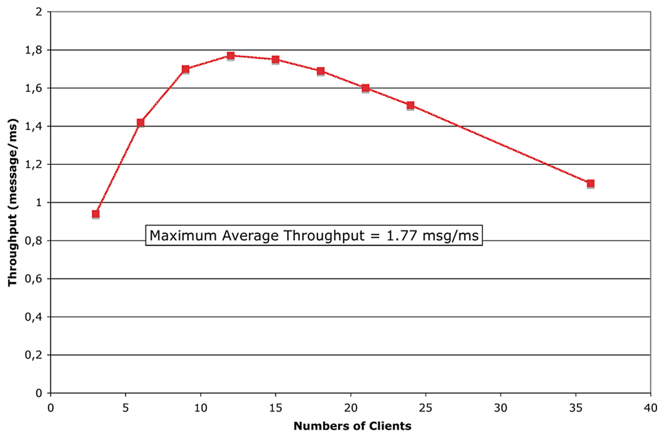


Fig. 5. Capacity of a stantard single queue

It is apparent that the increase in throughput is not a linear function of the number of producers and consumers. As well, when the maximum throughput is reached (with 4 producers), adding more producers and consumers can only reduce the average throughput.

Figure 6 presents the chart of the throughput and the numbers of messages waiting in the queue for the optimal setting for a single Joram queue. This optimal setting delivers the maximal throughput for a single queue of 1.77 message/ms. The throughput showed is stable at nearly 1.8 message/ms.

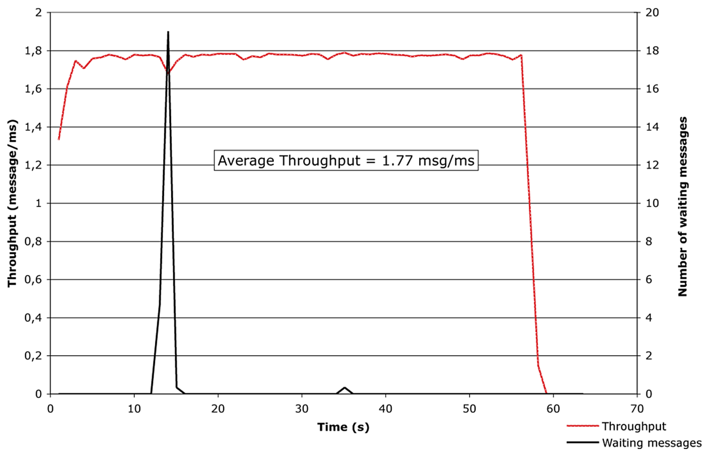


Fig. 6. Maximum Throughput of a Single Joram Queue

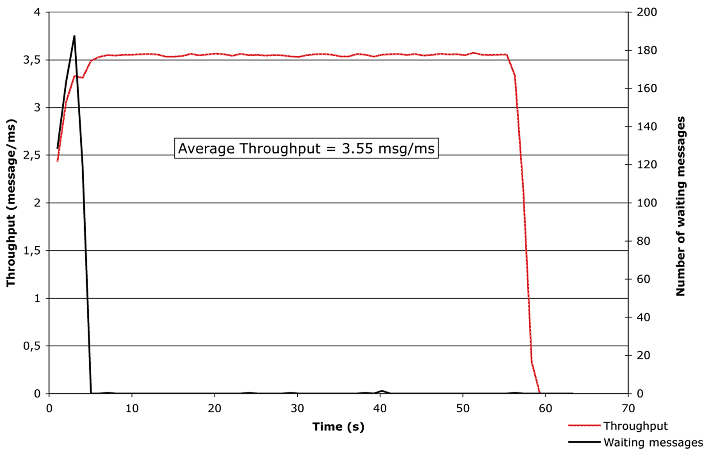


Fig. 7. Throughput of a stable queue cluster

Stable and balanced queue cluster. The goal of the next experiment was to find whether the increase in performance of a stable and properly balanced cluster queue was linear. In theory, a stable cluster queue should not exchange messages between the queues which are in the cluster. This experiment consisted of a cluster queue composed of 2 internal queues. On each queue, there were 4 producers and 8 consumers - *i.e.* the optimal configuration for the maximum throughput of a single queue.

Figure 7 shows the overall throughput and number of waiting messages of the cluster queue. The average throughput of the cluster queue (3.55 messages/ms) is about twice the maximum throughput of the single queue. The increase in throughput is linear and shows that the cost of managing a cluster without exchanging messages between the internal queues is negligible.

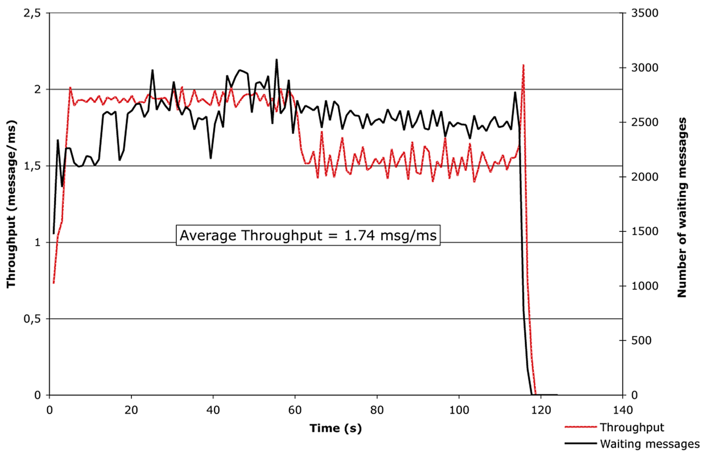


Fig. 8. Strong local instabilities in a queue cluster

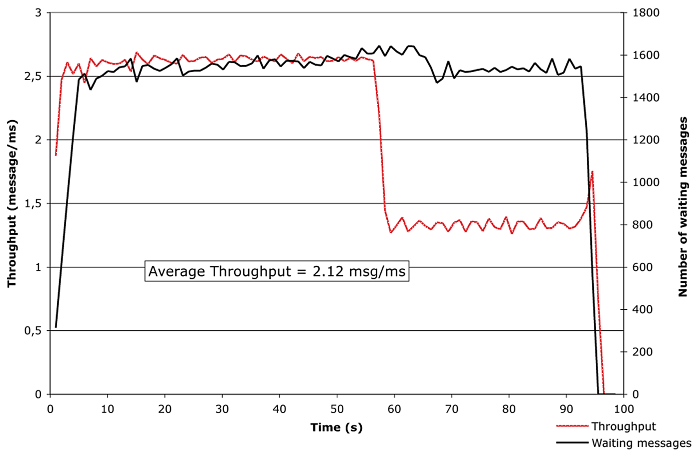


Fig. 9. Slight local instabilities in a queue cluster

Unbalanced cluster queue. The following figures demonstrate the strong impact of unbalance on the performance of a JORAM cluster queue. The same number of producers and consumers as the previous experiment were used but unbalance was introduced on the ratio of producer/consumer on the internal queues.

The experiment illustrated by Figure 8 had 7 producers and 2 consumers on the first internal queue and 1 producer and 14 consumers on the second one. The overall throughput shows a drastic decrease on the performance of the cluster queue. In fact, with an average throughput of 1.74 message/ms, it is better to use a single queue in this case. It would give a better throughput as well as costing less resources.

The instability is less pronounced in the experiment showed by the Figure 9. The first internal queue had 5 producers and 6 consumers. The second queue had 3 producers and 10 consumers. As can be seen on the chart, the overall throughput is only 2.12

messages/ms. It is much better than the previous one but it is still vastly inferior to the one presented in the Figure 7.

Conclusion for the measurements. These measurements show some interesting points. In a single queue, the critical factor impacting the performance is the number of messages waiting in the queue. Increasing the number of producers and consumers on a single queue leads to an increase in performance which is not linear. Furthermore a ceiling throughput is reached with (in our case) 4 producers and 8 consumers.

In a cluster queue, the balance of the cluster and the stability of the internal queues are extremely important. Even a slight instability between the queues strongly decreases the overall throughput. The instability seems to lead to an increase in the number of messages waiting in the queues. In contrast of a single queue, adding queues in a stable and well-balanced cluster leads to a linear increase in performance.

7.1 Algorithm Evaluation

We present here some results obtained by simulating the optimization algorithm. The aim is to demonstrate the efficiency of our algorithm in comparison to the original clients distribution scheme that is used in queue clusters.

The simulation runs a queue cluster composed of two queues Q_1 and Q_2 that share 40 message producers and 40 message consumers. The clients distribution is initially forced to the worst case: all producers are assigned to $Q1$ ($x_1 = 1$) while all consumers are assigned to Q_2 ($y_2 = 1$). Clients are configured to join and leave with equal probabilities, which ensure the global stability of the queue cluster.

Figure 10 presents the evolution of the clients distribution when using the original round-robin algorithm and Figure 11 shows the behaviour of the distribution when using the optimized algorithm. We observe that the original algorithm is unable to enforce a fair balancing of the clients: the unbalance is still roughly $0.3/0.7$ after 500 events, while our algorithm converges to the optimal distribution in less than 200 events. This concludes on the improvements expected by the use of the algorithm presented in this paper.

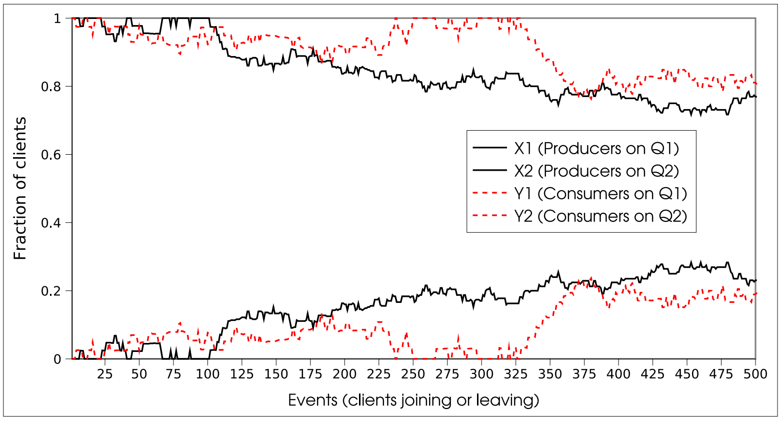


Fig. 10. Simulation with the original Round-Robin clients distribution algorithm

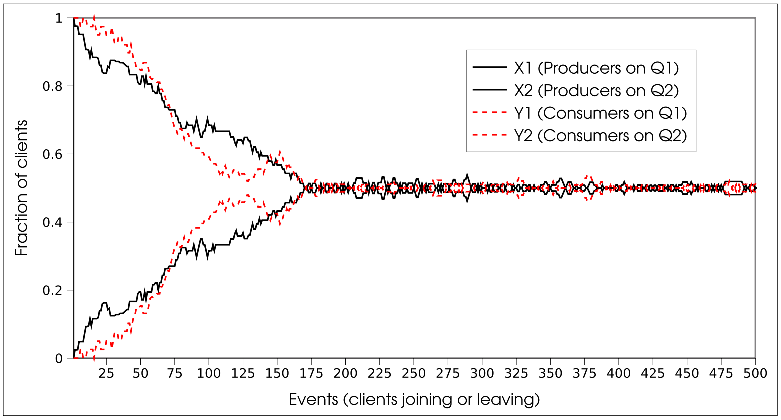


Fig. 11. Simulation with the optimizing clients distribution algorithm

8 Related Work

We describe a self-optimization mechanism in the case of a queue clustering technique. Some projects only analyse JMS performance whereas others target the self-optimization of J2EE infrastructure but do not focus on MOM self-optimization.

Regarding JMS performance, [1] provides an analysis of the throughput performance of JMS Using Websphere-MQ. [2] analyses a specific performance problem: The Message Waiting Time for the Fiorano-MQ Server. [3] describes a QoS Evaluation of JMS, it examines the impact of JMS attributes on performance.

About self-optimization, several projects which have addressed the issue of element management in a cluster of machines. In these projects, the software components required by any application are all installed and accessible on any machine in the cluster. Therefore, allocating additional resources to an application can be implemented at the level of the protocol that routes requests to the machines (Neptune [4] and DDSD [5]). Some of them (e.g. Cluster Reserves [6] or Sharc [7]) assume control over the CPU allocation on each machine, in order to provide strong guarantees on resource allocation.

9 Conclusion and Future Work

Providing a scalable and efficient Message Oriented Middleware is an important topic for today's computing environments. This paper analyses the performance of a Message Oriented Middleware and proposes a self-optimization algorithm to improve the efficiency of the MOM infrastructure. We describe (i) the key parameters impacting the performance of the MOM and (ii) the rules that control these parameters for optimal prformances. This paper also presents an evaluation that shows the impact of these parameters on the MOM.

Currently, the control loop has a very basic actuator to lead a client connection to a specific queue. The advantage of this actuator is its simplicity. However, the control loops cannot reconfigure the client connection during a session. Part of our future work is about providing a more powerful actuator. This actuator will provide the control loop with the ability to migrate a client connection when necessary. This requires a mechanim to move session data on other queue.

Acknoledgement. We would like to thank Sylvain Gonzalez (from the Sardes team) and André Freyssinet (from the Scal'Agent team) for their invaluable help. This work could not have been done without their support in setting up the experiments and providing insightful comments on the results.

References

1. Henjes, R., Menth, M., Zepfel, A.C.: Throughput performance of java messaging services using websphereMQ. In: DEBS. 5th International Workshop on Distributed Event-Based Systems, Lisboa, Portugal (July 2006)
2. Menth, M., Henjes, R.: Analysis of the message waiting time for the fioranoMQ JMS server. In: ICDCS. 26th International Conference on Distributed Computing Systems, Lisboa, Portugal (July 2006)
3. Chen, S., Greenfield, P.: Qos evaluation of jms: An empirical approach. In: HICSS 2004. Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9, p. 90276.2. IEEE Computer Society Press, Washington, DC, USA (2004)
4. Shen, K., Tang, H., Yang, T., Chu, L.: Integrated resource management for cluster-based internet services. In: OSDI-2002. 5th USENIX Symposium on Operating System Design and Implementation (December 2002)
5. Zhu, H., Ti, H., Yang, Y.: Demand-driven service differentiation in cluster-based network servers. In: INFOCOM-2001. 20th Annual Joint Conference of the IEEE Computer and Communication Societies, Anchorage, AL, IEEE Computer Society Press, Los Alamitos (2001)
6. Aron, M., Druschel, P., Zwaenepoel, W.: Cluster Reserves: a mechanism for resource management in cluster-based network servers. In: International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS-2000), Sant Clara, CA (2000)
7. Urgaonkar, B., Shenoy, P.: Sharc: Managing CPU and network bandwidth in shared clusters. IEEE Transactions on Parallel and Distributed Systems 15(1) (2004)

Minimal Traffic-Constrained Similarity-Based SOAP Multicast Routing Protocol

Khoi Anh Phan, Peter Bertok, Andrew Fry, and Caspar Ryan

RMIT University, School of Computer Science and Information Technology
GPO Box 3476V, Melbourne, VIC 3001, Australia
{thikhoi, pbertok}@cs.rmit.edu.au, andrew.fry@spotless.com.au,
caspar@cs.rmit.edu.au

Abstract. SOAP, a de-facto communication protocol of Web services, is popular for its interoperability across organisations. However, SOAP is based on XML and therefore inherits XML's disadvantage of having voluminous messages. When there are many transactions requesting similar server operations, using conventional SOAP unicast to send SOAP response messages can generate a very large amount of traffic [7]. This paper presents a traffic-constrained SMP routing protocol, called tc-SMP, which is an extension of our previous work on a similarity-based SOAP multicast protocol (SMP) [11]. Tc-SMP looks at the network optimization aspect of SMP and proposes alternative message delivery paths that minimize total network traffic. A tc-SMP algorithm, based on an *incremental* approach, is proposed and compared for its efficiency and performance advantages over SMP. Simple heuristic methods are also implemented to improve results. From extensive experiments, it is shown that tc-SMP achieves a minimum of 25% reduction in total network traffic compared to SMP with a trade-off of 10% increase in average response time. Compared to conventional unicast, bandwidth consumption can by reduced by up to 70% when using tc-SMP and 50% when using SMP.

1 Introduction

SOAP brings extensibility and interoperability to the communication and invocations of services among remote hosts. In contrast to its interoperability advantages, SOAP's major limitation is that its communication produces considerably more network traffic compared to its counterpart technologies such as CORBA and Java-RMI [15,6]. This issue has drawn great interest from many studies to propose techniques enhancing SOAP's performance. In the past, approaches for improving the network bandwidth performance of SOAP focused on the optimization of differential SOAP compression [15] and differential SOAP deserialization [1,13] techniques.

SOAP messages sent from the same server to multiple clients are generally have some similarity. It is important to emphasize that most SOAP messages have similar byte representations. SOAP messages created by the same implementation generally have the same message structure. Specifically, a SOAP message is surrounded by a large amount of non-domain-related XML data such as name space, encoding specifications and many other XML element names. Moreover, SOAP responses for the same SOAP request (with the same or different parameter values) will share the same XML document template. Even when the response messages are targeted for different operations of the same service, the response messages may have many similar data type structures.

Bandwidth is expensive in some environments such as mobile and wireless environments and sensor networks. At the same time, there has been an increased demand in the delivery of personalized information such as list of stock quotes, sports scores, weather forecasts and travel information to users, especially mobile users. If the number of receivers for a service is large and there is sufficient commonality in their interests, multicast will be an efficient way of delivering information. This is because network resources are used more efficiently by multicasts than broadcasts and replicated unicasts.

To reduce traffic caused by SOAP in low bandwidth environments, earlier we proposed a multicast technique, called SMP [11] (similarity-based SOAP multicast protocol), that is based on the similarity between SOAP messages and uses the shortest path first routing algorithm. SMP can reduce network traffic 47% more than SOAP unicast. To improve the performance, here we propose a new routing algorithm that can offer a higher reduction in network traffic than SMP.

We take an example of a real-time stock quote service to illustrate SMP's approach. The stock quote service allows users to request the prices and market information of a list of stocks. If there is news that will influence the price of a particular stock considerably, there would be enormous requests on that day for the stock's value. High numbers of requests will lead to high traffic volume. To avoid traffic congestion, SMP protocol can be used to replace unicasts because with SMP, instead of generating many messages with duplicated parts for different clients, the duplicated parts are sent only once for multiple clients — thus reducing the network traffic.

SMP relies on the Open Shortest Path First (OSPF) for routing which routes a message using the shortest path from a source to a destination. Therefore, the routing path selected by such an algorithm may not be the best in terms of traffic optimization. The more similar SOAP messages can be combined over specific links the more network bandwidth can be saved. However when OSPF is used, some SOAP messages that are very similar in content and follow the shortest paths, there may not be many common links shared by these messages. By restricting ourselves to the OSPF routing paths, we may miss other feasible paths that may result in further reduction in the total traffic. Using other paths may permit the aggregation of SOAP messages with a higher level of similarity and their transmission along more common links.

In this paper, we present an extension of SMP, called tc-SMP, which is a new routing algorithm for delivering SMP messages along paths such that there are more shared links between the deliveries of highly similar messages. Simulations have shown that tc-SMP can reduce network traffic, when compared to SMP, by more than 25% at the small cost of a 10% increase in response time. In addition, tc-SMP can reduce the traffic by up to 70% compared to unicast, and 40% compared to traditional multicast, the corresponding results for SMP are 60% and 30% respectively.

We will discuss SMP routing mechanisms and related work on QoS-based routing in the next two sections. Tc-SMP routing algorithm will be presented in Section 4. The performance experiments and results will be detailed in Section 5. We will then analyze the strengths and weaknesses of tc-SMP in the Discussion section. The paper will be concluded with final remarks and future work.

2 SMP Overview

We will present an overview of SMP's basic operations in this section to allow better understanding of tc-SMP. More details about SMP can be found in [11].

SMP was designed to deal with SOAP performance issues by exploiting the similar structure of SOAP messages. The goal is to reduce the total traffic generated over a network when sending SOAP responses from servers to clients. SMP allows similar SOAP messages that share some parts of the SOAP template to be sent in one customised SMP message instead of being sent as multiple copies.

Clients' addresses are represented as strings and stored in the SMP header, which is encapsulated inside the SOAP message body. The SMP body is also embedded inside the SOAP message body. There are two sections in the SMP body: (1) the <*Common*> section containing common values and structures of all messages addressed to clients encoded in the SMP header; (2) the <*Distinctive*> section containing individual different parts for each response message. The outermost envelope is referred to as an SMP message. The destination of an SMP message, which is specified in the SOAP header, is the next router in a network when the message is forwarded to all clients given in its SMP header.

Figure 1 explains the operations of SMP through a sample network of 1 source and 5 clients. Two SMP messages are sent out from the source s to its two next-hop routers r_1 and r_2. At r_2, the SMP message header is parsed to find out what clients the message is addressed to. The client addresses are then partitioned into a group of (c_4, c_5) and client c_3 alone based on the clients' next-hop routers. Since at r_2 the message targeting to client c_3 are sent independently, a unicast SOAP message, m_1, is used for responding to the *getStockQuote()* request. On the other hand, an SMP message targeting c_4 and c_5 is replicated from the incoming SMP message at r_2. In the new SMP message, the common part contains the content of the full m_2 message for the *getQuoteAndStatistic()* request and there is no distinctive section because both c_4 and c_5 request the same service operation.

Despite its advantage of saving network traffic, SMP has a disadvantage of using a conventional routing protocol (OSPF) to deliver messages to clients. Since OSPF uses Dijkstra's algorithm, SMP messages are routed along their shortest paths to destinations. Two nodes of a network are often connected with multiple paths. Therefore, sending messages just along least hop paths does not maximize the saving of traffic resulted from the similarity of messages.

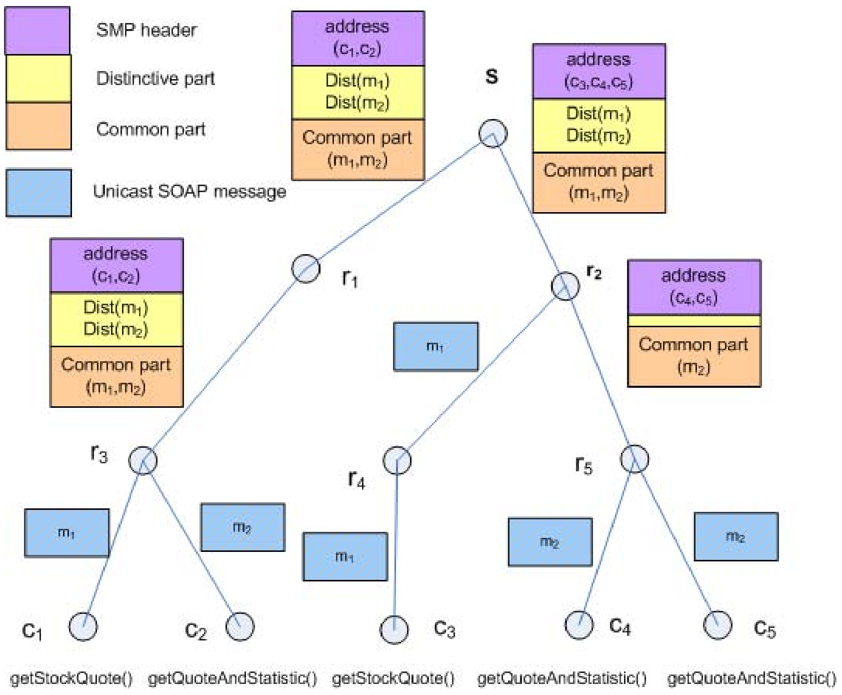


Fig. 1. A sample network illustrating how SMP routing works

In addition, SMP has a user-configured time frame. During this time period, outgoing SOAP response messages will be lined in a queue if their similarity level falls within a threshold limit. When a new request message arrives at the server, the server generates its corresponding SOAP response message and computes its similarity against existing on-queue messages. If the computed similarity satisfies the threshold then it is inserted into the queue. If not, the messages that already reside in the queue are sent out as an aggregated SMP message. As a result, the queue is empty for new requests and the above aggregation steps can be repeatedly carried out again. Messages in the queue can also be dispatched automatically after the defined time period.

It is important to note that to deploy SMP in a real network, all routers in the network need to be SMP-compatible. This can be done by installing an SMP software, which is an implementation of the proposed SMP, on each router to enable it to interpret SMP messages. The SOAP header in an SMP message specifies the next hop router as the message's destination. Therefore, when an intermediary router receives an SMP message, it processes the message as if it is the final destination of the message. Since an SMP-compatible router operates on the application layer, it has full access to the message's envelope and parses the SOAP body to get the list of clients encoded in the SMP header and the actual payload in the SMP body.

3 Related Work

In this section we present an overview of how routing trees are built in different QoS-based routing algorithms. Such information is important to understand the extension of SMP.

3.1 QoS-Based Routing Overview

The problem of finding a minimum cost multicast tree for sending similar SOAP messages can be categorized as a QoS-based routing problem. The main objective of QoS routing is to select paths that satisfy multiple QoS requirements in a network, while contributing to improved network resource utilization. QoS requirements are given as a set of constraints which can be link constraints, path constraints or tree constraints. Chen and Nahrstedt [4] define a *link constraint* as a restriction on link usage such as link bandwidth, while a *path constraint* is an end-to-end requirement on a single path such as end-to-end delay bounds or delay jitter. A *tree constraint* specifies a QoS requirement for an entire multicast tree, for example total traffic over the network or loss ratio. Resource utilization is often computed by an abstract cost metric [4]. The optimal QoS routing problem is then formulated as the lowest-cost path among all the feasible paths.

3.2 QoS Multicast Routing

Here we consider, in more detail, how a multicast tree which is rooted at a source and spans multiple destinations is actually built. There are several well-known multicast routing problems: Steiner tree and constrained Steiner tree problems. The *Steiner tree* problem (also called the *least cost* multicast routing problem) is to find the tree that spans a set of destinations with the minimum total cost over all links [8]. The *constrained Steiner tree* problem (often called the *delay-constrained least-cost* routing problem) is to find the least cost tree with bounded delay.

Multicast routing algorithms often belong to one of two categories: *source routing* and *distributed routing*. Source routing is a strategy that requires each node to maintain the global state of the network topology, as well as the link state information. Feasible paths to each destination are computed at the source node, based on the global state information. Routing decisions are then communicated to intermediate nodes via a control message. Contrasting to source routing, in distributed (or hop-by-hop) routing each node maintains state information for nearby nodes, exchanged via control messages. Each packet contains the address of its destination, and every node maintains a forwarding table that maps each destination address into transmitting-neighbor nodes. Forwarding decisions are made by each node, upon the receipt of a packet, after considering the destination of the packet but not its source. Some examples of QoS source and hop-by-hop routing algorithms are described in detail in subsequent sections.

The class of QoS multicast routing problems has been shown to have high computational complexity and to be NP-hard [16]. Hence these algorithms usually use heuristics to reduce the complexity of the path computation problem, at the expense of not achieving an optimal solution but just a feasible solution [10].

3.3 QoS-Constrained Multicast Source Routing

Some important work on source routing algorithms for multi-constrained multicasting is discussed. Chakraborty, et al. [3] proposed QoS-based Dynamic Multicast (MQ-DMR) algorithm which builds a dynamic multicast tree which satisfies multiple QoS requirements and efficiently distributes traffic throughout the network. It operates when adding a new node to the existing multicast tree. The two objectives of MQ-DMR are to minimize the overall network traffic when connecting a new receiver, and to connect a new node to the source so that QoS constrains are satisfied. A link cost is the inverse of the available bandwidth on that link. They aim to use part of the existing multicast tree to share the same path from the source to a newly added client, such that the additional traffic is minimized. A model is proposed to dynamically assign a cost to a link based on the duration for which it would be in use. MQ-DMR operates in a similar way to the Bellman-Ford shortest path algorithm [2]. In the first iteration, it finds one-hop least cost paths to all destinations. In the next iteration, two-hops least cost paths are found with a tendency to select links that are already in use in the multicast tree. Subsequent iterations are similar. The maximum number of iteration steps can be determined based on the pre-defined allowable delay jitter.

3.4 Hop-by-Hop Routing Algorithms

Though many proposed source routing algorithms (to solve QoS-constrained multicast routing problems) provide promising results, they have a common lack of scalability. This weakness stems from the fact that frequent updates of global states need to be made at the source to cope with the dynamics of the network and that means high computation overhead at the source [10]. To avoid this overhead, there have naturally been attempts to solve the QoS-constrained multicast problems in a distributed manner.

Shaikh and Shin [12] presented a destination-driven multicast routing algorithm (DDMC) that optimizes total tree cost. Most of the proposed heuristics that solve the minimum multicast tree problem assume the use of global cost information by the source, but DDMC uses only cost information from nearby nodes. The DDMC algorithm uses a greedy strategy based on Dijkstra's shortest path and Prim's minimal spanning tree algorithms. In DDMC, the costs at destination nodes are reset to zero to encourage the selection of paths that go through destination nodes. Any node reached from another destination node experiences only an incremental additional cost, thus the total tree cost can be reduced. However this method may result in a tree with long paths connecting multiple destination nodes, so it may not meet the end-to-end delay constraint. Additionally, in their tree model, cost is not associated with any specific network parameter. However different cost metrics may have different meanings and implications on the overall performance of an algorithm. Shaikh and Shin did not provide results about the algorithm's performance on different cost metrics such as link capacity, hop distance, inverse of link bandwidth, or congestion rate; which makes it hard to have a clear idea about the advantage of their approach.

4 Traffic-Constrained SMP Routing

This section formally defines the optimization problem that the tc-SMP algorithm aims to solve and outlines preliminary models and concepts.

4.1 Problem Definition and Notations

The objective of tc-SMP is to find routing paths linking the source to a set of clients while simultaneously minimizing the total traffic cost (related to the construction of the tc-SMP tree). We formulate this problem as the *traffic-constrained similarity-based SOAP multicast* routing problem (tc-SMP). A graph based formulation of tc-SMP is given here.

Let $G(V, E)$ be a network graph, where V is the set of nodes and E is the set of edges, s is the source node, $C = \{c_1, c_2, c_3, ...c_N\}$ is the set of clients (or destinations) in the network, so $s \cup C \subseteq V$. A tree $T(s, C) \subseteq G$ originates at s and spans all members in C. Let $P_T(c_i) \subseteq T$ be the set of links in T that constitutes the path from s to client $c_i \in C$. $tr(e), e \in E$ is a function that gives the number of bytes transmitted through link e. The total traffic generated in the tree T, denoted by $\mathit{Traffic}(T)$, is defined as the sum of the traffic transmitted on all links in the tree, given by the expression $\mathit{Traffic}[T] = \sum_{e \in T} tr(e)$. The objective of the tc-SMP problem is to construct an tc-SMP tree $T(s, C)$ such that the tree's network traffic, $\mathit{Traffic}[T]$, is minimized.

The traditional multicast routing tree problem (which sends *identical* messages to multiple receivers from a source while simultaneously minimizing some cost function) is an NP-hard problem [9]. The tc-SMP problem is also NP-hard.

Definitions of a tc-SMP routing tree and a branching node, used in the tc-SMP algorithms, are defined now.

Definition 1. *[tc-SMP Tree]: A tc-SMP tree is a tree of tc-SMP nodes. A tc-SMP r_j node is a data structure in the form of [$r_j.router, r_j.clients, r_j.cost$] where:*

1. *$r_j.router$: corresponds to a physical router in the network topology,*
2. *$r_j.clients$: a set of clients that this node forwards SMP messages to, and*
3. *$r_j.cost$: total traffic already generated in the network when the SMP message arrives at r_j.*

Definition 2. *[Branching Node]: A node B is a branching node of a client C with respect to an tc-SMP tree T if the following conditions are met:*

- *$B \in T$.*
- *C is connected to the tree through B and the connection introduces minimal additional traffic to the total tree cost.*

4.2 The Algorithm

The incremental tc-SMP algorithm iteratively examines a branch connecting to each client in the SMP tree and determines if the branch can be replaced by a substitute path in the network to reduce traffic load created when transmitting a message to the client. If such a substitute branch does not exist, the shortest path to the client from the SMP tree will be added to the tc-SMP tree. Therefore, in the worst scenario, incremental tc-SMP performs as well as SMP.

The following notations and assumptions are made for the proposed *incremental improvement* tc-SMP routing algorithm:

- Let T_{tc-SMP} be the set of routers in the tc-SMP spanning tree.
- Let T_{temp} be a temporary tree built during the path finding process for each client.
- Let T_i be a tree resulted after each iteration of adding a new client to the tc-SMP spanning tree.
- Let $C_K = \{c_1, c_2, c_3, ..., c_K\}$ be a group of clients that have requested for $|M_K|$ SOAP messages that have a similarity level greater or equal to the required threshold and can be aggregated into one SMP message.
- Let $M_K = \{m_1, m_2, m_3, ..., m_K\}$ be the set of SOAP response messages to be sent to all clients. m_i is the response message for client c_i.
- Let $Cost(T)$ be a function to compute the total traffic cost of the whole tree T based on maximizing similarity.

Algorithm 1 shows various steps for incrementally building a tc-SMP tree. The steps can be grouped into three main phases as presented in the following list. The details of each phase are elaborated in the subsequent paragraphs.

1. **Phase 1:** Setting up an SMP tree based on shortest paths (Line 1 of Algorithm 1).
2. **Phase 2:** Finding alternative paths connecting to each client (Lines 15-16 of Algorithm 1).
3. **Phase 3:** Building a temporary tree, T_{temp}, which includes the newly found alternative path to the client (Lines 18-19 of Algorithm 1 and Algorithm 2).
4. **Phase 4:** Selecting the branch that connects to the client with the least traffic to add to the tc-SMP tree (Lines 20-23 of Algorithm 1).

In the first phase, the source establishes a routing tree, called T_{SMP}, where every path from the source to each client is based on Dijkstra's shortest path first algorithm as being done in the original SMP algorithm. This phase can easily be completed by using the OSPF routing protocol deployed in most networks. Each node r_j in the T_{SMP} also has the properties $r_j.router$, $r_j.clients$ and $r_j.cost$ as defined in Definition 1.

The tc-SMP tree, T_{tc-SMP}, is initially empty. Each client is added to the tc-SMP tree one after another. In the first iteration, among K clients, a random client c_{rand} is chosen as the first client to be added to the T_{tc-SMP}. The first

Algorithm 1. Incremental tc-SMP algorithm

```
1  $T_{SMP} = \{r_k, r_{k+1}, ..\}$: an SMP tree
2  $T_{temp} \leftarrow T_{SMP}$; $T_{tc-SMP} \leftarrow \emptyset$
3  foreach $c_i \in C_K$ do
4      if $T_{tc-SMP} = \emptyset$ then
5          foreach $r_j \in p_{s,c_i}$ do
6              {$p_{s,c_i}$ is the shortest path from $s$ to $c_i$ }
7              $T_{tc-SMP} \leftarrow T_{tc-SMP} \cup r_j$
8          $T_i \leftarrow T_{SMP}$
9          continue
10     else
11         {$p^{min}_{s,c_i}$ is a path from $s$ to $c_i$ that introduces the least traffic in the tree}
12         $p^{min}_{s,c_i} \leftarrow p_{s,c_i}$ , where $p_{s,c_i} \in T_{i-1}$
13         $T_i \leftarrow T_{i-1}$
14         foreach $r_j \in T_{tc-SMP}$ do
15             if $r_j \neq s$ AND $r_j \notin C_k$ AND $r_j \notin p_{s,c_i}$ then
16                 $\mathsf{R} = \mathtt{FindSP}(r_j, c_i)$
17                 if $\mathsf{R} \neq \emptyset$ then
18                     $T_{temp} \leftarrow \mathtt{RemoveClientFromTree}(T_{temp}, c_i)$
19                     $T_{temp} \leftarrow T_{temp} \cup \mathsf{R} \cup\ p_{s,r_j}$
20                     if $\mathtt{Cost}(T_{temp}) < \mathtt{Cost}(T_i)$ then
21                         $p^{min}_{s,c_i} \leftarrow p_{s,r_j} \cup p_{r_j,c_i}$
22                         $T_i \leftarrow T_{temp}$
23         $T_{tc-SMP} \leftarrow T_{tc-SMP} \cup p^{min}_{s,c_i}$
```

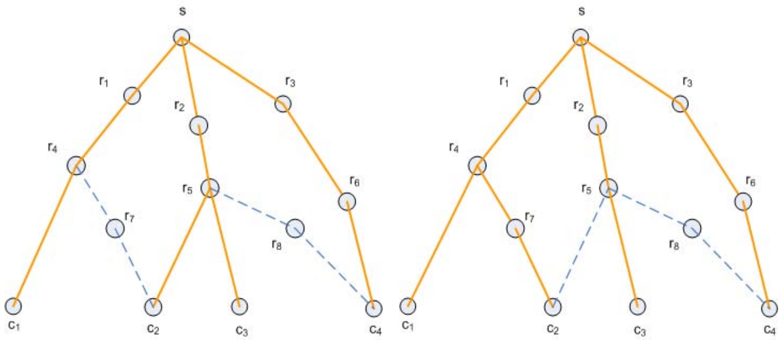
client is a special case where the tc-SMP path to c_{rand} is the same as the SMP path to c_{rand} (see Lines 4–9 of Algorithm 1).

In subsequent iterations, the algorithm attempts to find an alternative path for another client, say c_i, as illustrated in Lines 12–23 of Algorithm 1). We denote T_{temp} a temporary tree which is built during the process of finding an alternative path for client c_i. The tc-SMP algorithm examines if there exists a path from a node (excluding the source and the clients), r_j, that is already in T_{tc-SMP} for c_{i-1}, to the next client c_i. Also, the algorithm ignores r_j if it resides on the original shortest path p_{s,c_i} from the previous tree.

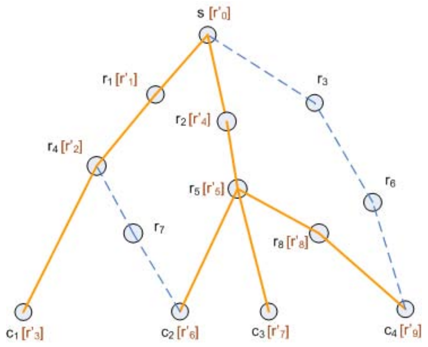
If there is a path from r_j to c_i that meets the requirement in Line 15 of the main algorithm, the T_{temp} is then built based on T_{i-1}, the resulted tree from the previous iteration, but excluding the path spanning to c_i (see Line 18 of Algorithm 1 and Algorithm 2). Subsequently, the path p_{r_j,c_i} is added to T_{temp}. The total cost of T_{temp} is then calculated as a result of this and compared to the T_{i-1}'s cost. If $Cost(T_{i-1})$ is greater than $Cost(T_{temp})$, T_i will be assigned to T_{temp}. This process continues until all the clients in the list are examined for alternative paths.

Algorithm 2. *RemoveClientFromTree* Procedure: Remove nodes spanning to a client from a tree

```
Input: $T$: the input tree
c: The client to be removed from $T$
Output: $T$: The resulted tree after removing c
1 procedure RemoveClientFromTree do
2     foreach $r_i \in p_{s,c}$ do
3         if $|r_i.clients| = 1$ then
4             {There is only c passing through this router }
5             $T \leftarrow T \setminus r_i$
```



(a) The initial SMP tree (b) A temporary tree



(c) The final tc-SMP tree

Fig. 2. The initial SMP tree, a temporary tree and the final tc-SMP tree built by the incremental tc-SMP algorithm in a sample network

4.3 Example Illustration

The diagrams in Figure 2 show an example of how the incremental tc-SMP algorithm operates. In this example, there are 4 clients c_1, c_2, c_3 and c_4 all requesting the same SOAP message. Let ω denote the size of each SOAP response message. For simplicity, the size of an aggregated SMP message comprising any number of original SOAP messages (based on similarity) is still ω[1]. First, an SMP tree, denoted by T_{SMP}, is built by using the OSPF protocol. T_{SMP} roots at the source s and spans all four clients on the shortest paths, as shown in solid lines in Figure 2(a). The cost in terms of the total traffic created if SMP messages are sent following the paths in T_{SMP} is then computed ($Cost(T_{SMP}) = 10\omega$). Next, a tc-SMP tree, denoted by T_{tc-SMP}, will be built gradually by adding the clients to the tree one after another. The first client, c_1, is added to the tc-SMP tree through the shortest path, therefore, the source s, routers r_1 and r_4 and client c_1 are included in the T_{tc-SMP} tree.

To add client c_2 to T_{tc-SMP}, the algorithm examines if there exists a path from a node, that resides in T_{tc-SMP} and satisfies the criteria on Line 15 of Algorithm 1, to c_2. Routers r_1 and r_4 both have paths to c_2. Let us consider r_4 as the branching node for linking c_2 as r_4 is closer to c_2 than r_1 is. A temporary tree, denoted by T_{temp}, is built by removing the single path spanning c_2 in the T_{SMP} tree and adding the new path connecting r_4 to c_2. This T_{temp} tree is depicted in Figure 2(b) (in solid lines). The cost of this temporary tree is computed and equal to 11ω which is higher than the cost of the initial T_{SMP} tree (10ω). Therefore, in this iteration the shortest path from the source to client c_2 is added to T_{tc-SMP}.

Client c_3 can be added to T_{tc-SMP} easily because there is only one path from the source to c_3. There are two paths spanning client c_4, and it is trivial to realize that c_4 will be added to T_{tc-SMP} via r_5. The final incremental tc-SMP tree is illustrated in Figure 2(c) along solid lines.

4.4 Heuristic Methods

We have described the basic functionalities of the two tc-SMP routing algorithms without using any heuristics. Here two simple methods, which can be applied to Algorithm 1 to determine the order of clients in the distribution list to be added to the tree, are proposed.

- **Message size-based Heuristic Approach** (MHA): This approach is based on the sizes of the response messages. First the K SOAP response messages are sorted in a descending order according to their sizes. Clients are added to the tree in order of their descending message size. Using this method, large messages are sent out first along the least hop paths, thus less traffic is generated.
- **Similarity-based Heuristic Approach** (SHA): This heuristic method is based on the similarities between the response messages. The first client to

[1] In practice, additional several bytes are required for storing clients' addresses.

be added to the tree is the one that has the largest message size. Then subsequent clients are added to the tree in order of descending message similarity with existing clients' messages in the tree. With this method, messages with higher similarity tend to be sent along more shared links, thus more network bandwidth can be saved.

4.5 Complexity Analysis

We will analyze the time complexity of the proposed algorithm to build the tc-SMP routing tree. We show that the computation time for the algorithm is polynomial by proving the following theorem.

Lemma 1. *The time complexity of the incremental algorithm to build a traffic-constrained SMP routing tree is* $O(n(m+nlogn))$ *where* m *is the number of edges and* n *is the number of nodes in a network.*

Proof. The worst case for this algorithm is when building a tc-SMP tree that spans all N clients. Considering the main for-loop, Lines 3–23 of Algorithm 1. This *for-loop* is executed once for each client and hence a total of N times. As explained above, N is considered as a constant in this analysis as there is often an upper bound on the number of clients that can be aggregated in an outgoing SMP message from the server. Inside this loop, the algorithm runs through all the nodes that already exist in the tc-SMP tree, which in the worst case would have n nodes. Finding the shortest path from a node to a client (as presented in Line 16 of Algorithm 1) requires $O(m + nlogn)$ time using a Fibonacci heap implementation [5]. Line 18 calls the $RemoveClientFromTree$ procedure, described in Algorithm 2, which requires $O(L)$ time where L is the largest number of hops from the source to any client in the network. The time complexity required to measure the cost of a temporary tree, Line 20, is $O(n)$. Therefore, the *for-loop* of Lines 3–23 takes $O(n(m+nlogn)+L+n)$, simplified to $O(n(m + nlogn))$, to complete. In conclusion, execution time of the *incremental* tc-SMP algorithm is of the order $O(n(m + nlogn))$ time.

The proposed algorithm is more complex than the OSPF algorithm because the OSPF problem finds the optimal path for each destination independently. In contrast, our traffic-constrained SMP problem involves path optimizations for multiple destinations the paths of which are dependent on each other and there is a common constraint of reducing overall network traffic.

5 Tc-SMP Performance Evaluation

To test the effectiveness of tc-SMP over SMP, we evaluated its performance and compared it to SMP, traditional multicast and unicast communications. Details of the experimental setup and simulation results are described in this section.

5.1 Experimental Setup

We used OMNeT++ [14] as the simulation program to randomly generate different hierarchical network topologies to carry out our experiments. In these topologies, the maximum number of hops for the shortest paths from the source to any client is 10. The propagation delay, which is the time that a network message takes to travel from one node to another is constant at $t_{prop} = 5ms$. The topologies generated such that there were always multiple paths to route a message from the source to most of the clients. The number of clients in the network ranged from 10 to 200. For each network topology, we performed six tests: incremental tc-SMP without heuristic, incremental tc-SMP with the similarity based heuristic, incremental tc-SMP with the message-size based heuristic, SMP, traditional multicast and unicast. For each test, 20 experimental runs were performed and the result given are the average of these runs.

Clients make requests to the Web service operations followed a Zipf distribution [17] with a skewness factor of $\alpha = 1$. In our experiments, there were 10 operations defined in the Web service's description document. These operations correspond to 10 SOAP response messages (denoted by $m_1, m_2, \ldots, m_{10}$) in which m_1 is the most frequently requested message and m_i is the i^{th} frequently accessed one. The size of the messages ranged from 20Kb to 50Kb. The similarity threshold used for SMP and tc-SMP methods is 0.6. The similarity between messages depends on requested service operation and its input parameters. The simulated bandwidth available on each link was 1.5Mbps.

5.2 Experimental Results

Total network traffic and average response time for each client are the two metrics used to examine the performance of the tc-SMP algorithm and to compare with SMP, multicast and unicast. The network load is the total size of all messages passing through all links in the routing tree when sending responses to all the clients. The average response time is the average time it takes from when the server to send a response message out until the message reaches the destined client. It is computed by dividing the sum of the delays that each client experiences by the number of clients. The response time includes propagation and transmission delays on each link and processing delays at the server and at intermediary nodes.

A) Total Network Traffic: Figure 3 shows the total network traffic for the incremental tc-SMP algorithm compared to SMP, multicast and unicast schemes. As expected, unicast produces the greatest volume of traffic, that is proportional to the number of receivers. Traditional multicast protocol represents an improvement of around 30% over unicast, while SMP and tc-SMP can reduce traffic by up to 50% and 65% respectively. With a small network of under 50 clients, the reduction in traffic between tc-SMP and SMP over unicast are quite small, with little difference between them (around 15%). With larger networks (100 to 200 clients) tc-SMP's and SMP's performance gain over unicast in traffic is more significant — over 60% for tc-SMP and over 45% for SMP. Comparing tc-SMP

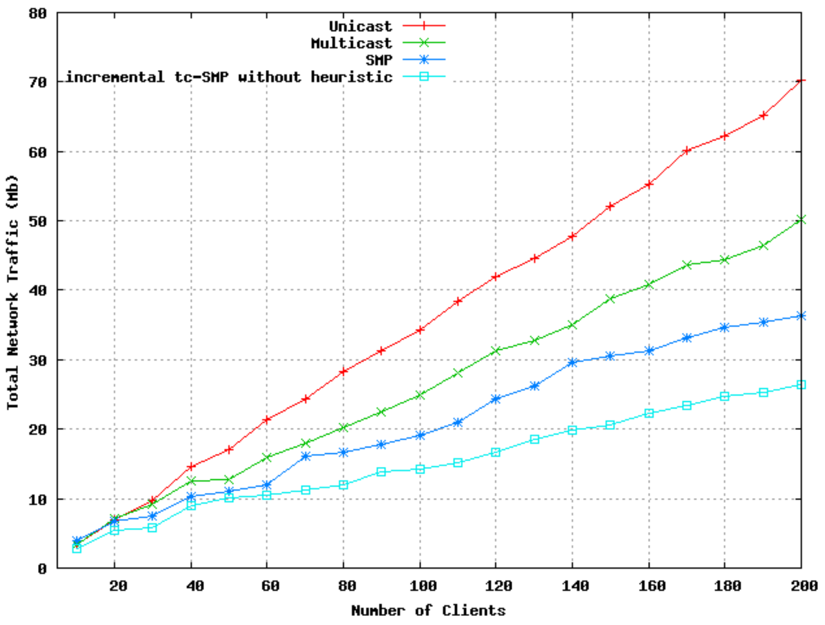


Fig. 3. Total network traffic comparisons between different routing protocols

to SMP, the difference in bandwidth consumption is not noticeable with small networks of 10 or 20 clients. When the client numbers increase to 50, 100 and 200, tc-SMP outperforms SMP by around 10%, 20%, and 25% respectively.

Figure 4 compares the total network traffic for the tc-SMP algorithm with and without heuristics. In general, there is no significant difference between the network traffic results with or without tc-SMP heuristics. A close look reveals that the similarity-based heuristic method presents an improvement of approximately 3% over its message size-based heuristic method counterpart. For example, with a network of 150 clients, the incremental tc-SMP algorithm with message-size based heuristic and similarity based heuristic generate 21.9Mb and 19.23Mb traffic respectively. It is evident that the incremental tc-SMP algorithm with the similarity-based heuristic method produces least traffic of the tc-SMP variations by a margin of around 5%.

B) Average Response Time: The average response times observed in the experiment is shown in Figure 5. The unicast method has the lowest average response time at approximately 59ms for networks with 10 clients and 116ms for networks with 200 clients. The traditional multicast protocol is about 1.5 times slower than unicast. SMP performs slightly slower than multicast with about 10% higher average response time. The response time for the tc-SMP method is about 2.0 to 2.5 times higher than the unicast method. This represents an average increase of 15% in response time over multicast.

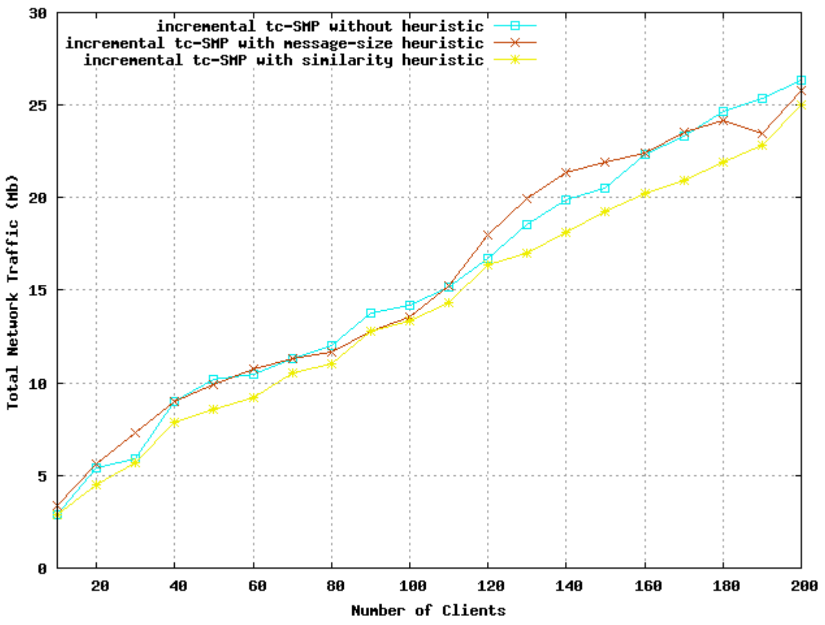


Fig. 4. Total network traffic comparisons between different heuristics and non-heuristic tc-SMP algorithms

SMP and tc-SMP have significant processing overhead at the server required to measure the similarity between messages and to aggregate the similar ones. Small additional processing time at intermediate nodes is required because mid-way routers need to split incoming SMP messages into multiple outgoing messages for next-hop routers. Similar overhead also occurs in traditional multicast routing but is slightly smaller at transitional routers.

The performance penalty of tc-SMP over SMP is primarily its overhead in setting up the routing tree initially at the server. However, the difference in the average delays is not significant. For tc-SMP without heuristic, the average delay a client experiences ranges from 60ms to 195ms for networks of 10 clients to 200 clients. The corresponding results for SMP are 59.5ms to 175ms. On average, using the tc-SMP algorithm raises the average response by around 10% compared to SMP.

Figure 6 shows the response times for the tc-SMP algorithms with and without heuristics. Between the two heuristics used for selecting the order in which clients are added to a tc-SMP tree, the method that is based on the similarity between response messages takes a longer time. In the similarity-based heuristic method, the largest message is found first, then subsequent messages that have the greatest similarity with the first message will join a tc-SMP tree. Therefore, it is reasonable to expect that the similarity based heuristic will have higher response time than the message-size based heuristic as observed in Figure 6.

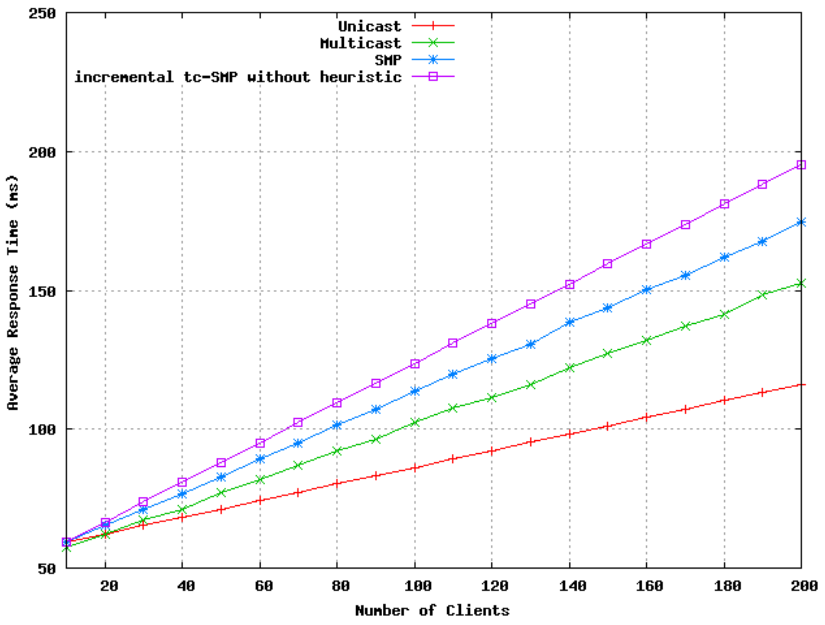


Fig. 5. Average response time comparisons between different routing protocols

6 Discussion

Using tc-SMP, the traffic load is close to 4 or 5 times smaller than the traffic generated when the individual original SOAP response messages are sent as unicasts. Tc-SMP reduces network bandwidth consumption of around 30 percent compared to SMP. A disadvantage of tc-SMP over SMP is that it requires additional time to build the routing tree, which leads to an average response time increase of less than 10 percent.

As shown in Section 4.5, the tc-SMP routing algorithms can be performed in polynomial time, so the additional computation time is acceptable. The use of tc-SMP can be justified by traffic reduction whenever the increased response time is acceptable — from 3.5 up to 5 times reduction in traffic compared to under 2.5 times increase in average response time. Of course, results vary depending on the configuration of the network.

This amount of delay is tolerable for many Web service applications, for example wireless communication among Intranet users, and personalized information retrieval over mobile networks. Tc-SMP represents a method for compressing size of messages in a network, thus it may be suitable for sensors network applications where bandwidth is limited and devices are constrained in power and battery life. Reducing the bandwidth consumption also benefits other applications by reducing traffic which is sent across those same links. The tc-SMP algorithm is

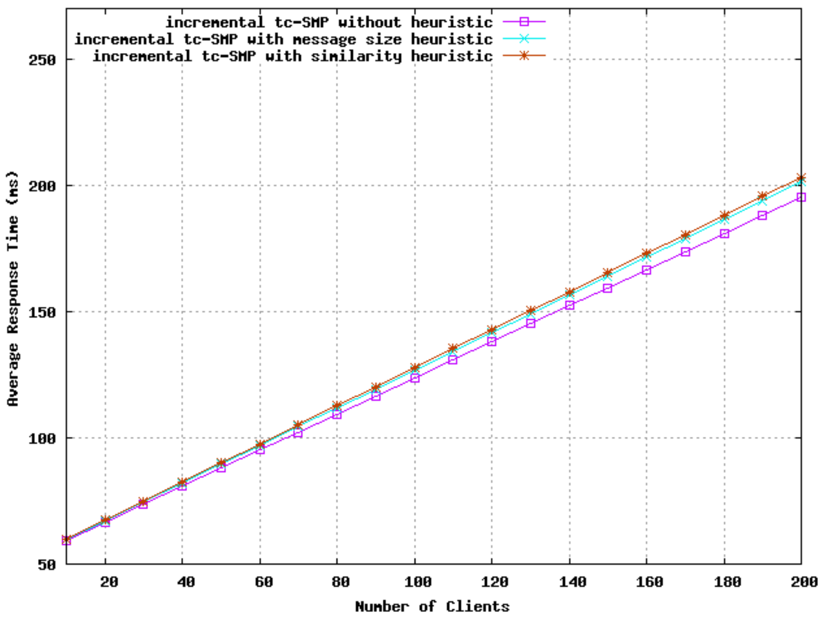


Fig. 6. Average response time comparisons between different heuristics and non-heuristic tc-SMP algorithms

particularly suitable in cases where the underlying networks are known to have multiple paths between nodes.

In addition, in high-speed wide-area networks, the link transmission delay is usually in microseconds, while the propagation delay may be close to milliseconds (much more greater) — so given a powerful processor, the trade-off in higher server processing overhead using tc-SMP over SMP is negligible. In wide-area networks propagation delay contributes the majority of the total delay.

7 Conclusion and Future Work

This paper outlined a method to reduce SOAP traffic in low bandwidth networks by using similarity-based multicasting. Tc-SMP represents an improvement over SMP, an earlier proposed algorithm, for SOAP multicast traffic. Tc-SMP is different from SMP in a way that a new source-routing algorithm is used in tc-SMP for delivering aggregated messages along paths to introduce the minimal traffic in the network, instead of using the OSPF routing method which is widely used on the Internet.

The problem of building a traffic-constrained similarity-based SOAP multicast tree is NP-complete. The proposed algorithm provides a good solution and operate in polynomial time with a complexity of $O(n(m + nlogn))$. The algorithm is based on sending combined messages to clients along shared paths by selecting a path to each client that introduces minimal cost increment.

Simulations have proven that tc-SMP algorithm reduces traffic generated in the network even further by around 30% when compared to SMP. The performance trade-off of tc-SMP over SMP is an increase of less than 10% in average response time. As the number of clients increases, the network traffic caused by tc-SMP is considerably less than that caused by SMP, while the performance penalty is comparatively small. We also implemented a heuristic method for the tc-SMP algorithm. The heuristic based on the similarity between messages to new clients and messages to existing clients gives a gain of around 3% over tc-SMP without any heuristic. The heuristic based on message size achieves negligible performance gain.

Future work will involve in researching better heuristic algorithms to further improve performance. Considerations of other quality of service parameters such as delay bounds and bandwidth requirements for each client may also be incorporated into the tc-SMP routing algorithm.

Acknowledgment

The authors would like to thank the Australian Research Council for supporting this project, entitled "Designing an Efficient and Scalable Infrastructure for Mobile Web Services." under grant No. LP0455234.

References

1. Abu-Ghazaleh, N., Lewis, M.: Differential deserialization for optimized soap performance. In: Proceedings of the 2005 ACM/IEEE Conference on Super-Computing, Seattle, WA, USA, pp. 21–31 (November 2005)
2. Bellman, R.: On a routing problem. Quarterly of Applied Mathematics 16(1), 87–90 (1958)
3. Chakraborty, D., Chakraborty, G., Shiratori, N.: A dynamic multicast routing satisfying multiple QoS constraints. International Journal of Network Management 13(5), 321–335 (2003)
4. Chen, S., Nahrstedt, K.: An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. IEEE Networks Magazine, Special Issue on Transmission and Distribution of Digital Video 12(6), 64–79 (1998)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, vol. 2, pp. 595–601. MIT Press and McGraw-Hill (2001)
6. Elfwing, R., Paulsson, U., Lundberg, L.: Performance of SOAP in Web service environment compared to CORBA. In: Proceedings of the 9th Asia-Pacific Software Engineering Conference, Gold Coast, Australia, pp. 84–94. IEEE Computer Society Press, Los Alamitos (2002)
7. Govindaraju, M., Slominski, A., Chiu, K., Liu, P., van Engelen, R., Lewis, M.J.: Toward characterizing the performance of SOAP toolkits. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA, pp. 365–372. IEEE Computer Society, Los Alamitos (2004)
8. Hwang, F., Richards, D., Winter, P.: The Steiner Tree Problem. Elsevier, North-Holland (1992)

9. Oliveira, C., Pardalos, P., Resende, M.: Optimization problems in multicast tree construction. In: Handbook of Optimization in Telecommunications, pp. 701–733. Kluwer, Dordrecht (2005)
10. Paul, P., Raghavan, S.: Survey of QoS routing. In: Proceedings of the 15th International Conference on Computer Communication, Mumbai, India, pp. 50–75. International Council for Computer Communication (August 2002)
11. Phan, K.A., Tari, Z., Bertok, P.: Optimizing Web services performance by using similarity-based multicast protocol. In: Proceedings of the 4th European Conference on Web Services, Zurich, Switzerland, pp. 119–128 (December 2006)
12. Shaikh, A., Shin, K.: Destination-driven routing for low-cost multicast. IEEE Journal of Selected Areas in Communications 15(3), 373–381 (1997)
13. Suzumura, T., Takase, T., Tatsubori, M.: Optimizing Web services performance by differential deserialization. In: Proceedings of the IEEE International Conference on Web Services, Orlando, Florida, USA, pp. 185–192. IEEE Computer Society Press, Los Alamitos (2005)
14. Varga, A.: OMNet++ Discrete Event Simulation System, URL (2006), http://www.omnetpp.org
15. Werner, C., Buschmann, C., Fischer, F.: WSDL-driven SOAP compression. International Journal of Web Services Research 2(1), 18–35 (2005)
16. Yuan, X.: Heuristics algorthims for multiconstrained quality-of-service routing. IEEE/ACM Transactions on Networking 10(2), 244–256 (2002)
17. Zipf, G.K.: Human Behaviour and the Principle of Least-Effort. Addison-Wesley, Cambridge MA (1949)

Implementing a State-Based Application Using Web Objects in XML

Carlos R. Jaimez González and Simon M. Lucas

Department of Computer Science, University of Essex,
Wivenhoe Park, Colchester CO4 3SQ, UK
{crjaim,sml}@essex.ac.uk

Abstract. In this paper we introduce Web Objects in XML (WOX) as a web protocol for distributed objects, which uses HTTP as its transport protocol and XML as its format representation. It allows remote method invocations on web objects, and remote procedure calls on exposed web services. WOX uses URIs to represent object references, inspired by the principles of the representational state transfer (REST) architectural style. Using URIs in this way allows parameters to be passed, and values returned, either by value or by reference. We present a case study, in which an existing chart application is exposed over the Internet using three different technologies: RMI, SOAP, and WOX. WOX proves to be the simplest way to implement this application, requiring less program code to be written or modified than RMI or SOAP. Furthermore, as a consequence of its REST foundations, WOX is particularly transparent, since any objects that persist after a WOX call may be inspected with any XML-aware web browser. It is also possible to invoke methods of persistent objects through a web browser.

1 Introduction

Exposing applications over the Internet has become essential for many areas, due to the potential advantages of accessing data and objects from any place in the world. For this purpose, there are many existing distributed object technologies such as the Remote Method Invocation (RMI) [6] and the Common Object Request Broker Architecture (CORBA) [2], and web service technologies such as XML-RPC [7] and the Simple Object Access Protocol (SOAP) [5]. Both distributed object and web service technologies allow applications to be remotely accessible and allow more complex systems to be composed of components residing on geographically distributed machines. There are, however, some important differences between these two technologies, which can affect their suitability for specific types of applications.

Distributed object technologies base their functionality on two concepts: the *object's reference*, which allows a client application to refer to an existing object and execute operations on it; and the *object's state*, which is maintained between operation calls that can modify it. On the other hand, web service technologies in their current state do not have any of the concepts of distributed object programming and consequently have significant limitations. They do not support access to remote objects, but instead they provide standalone services through the web by exchanging eXtensible Markup Language (XML) [3] messages, and attempt to solve the interoperability problem that exists with distributed object technologies. Although existing distribute object systems such as CORBA and RMI can work on the web by tunneling their requests through HTTP, this decreases their performance, and can be a technically demanding task.

The suitability of every technology to implement a given application and expose it over the web depends on a series of aspects that need to be considered, such as the importance of interoperability between languages, the encoding of messages, the maintenance of object references (object state), the efficiency in the transfer of data, the support and extensibility of data types, the ease of use and implementation, among others. To explore the issues, we present a case study based on a simple state based data charting application, which was originally implemented in Java. The original application is stateful, and allows data to be incrementally added to a chart object. A user creates a chart object with a title, and can then successively add data-points; this incremental style makes the chart application very easy to use for a client program. For our case study, the application needs to be exposed over the Internet, preserve the state of objects, and have access to remote objects. The use of XML as the encoding format for objects provides a simple text representation of any kind of data, which is machine and human readable, and it also provides a standard format to transfer data, which could lead to language independence. With the existing distributed object and web service technologies, it would be possible to implement such an application, but considerable extra programming effort would be required.

Web Objects in XML (WOX) is a web distributed object protocol that offers features from distributed object programming and web services to expose applications over the Internet. WOX allows the creation of remote objects, the invocation of methods on remote web objects and the invocation of remote procedure calls on exposed web services, among other operations. WOX uses HTTP as its transport protocol, XML as its format representation, and makes objects available through their own Uniform Resource Identifier (URI) [8], inspired by the principles of the Representational State Transfer (REST) [9] architectural style. WOX is simple and light-weight.

This paper is organized as follows. Section 2 introduces the WOX architecture, the set of client operations supported, and the format of the XML messages exchanged. A case study is presented in section 3, in which RMI, SOAP and WOX are evaluated to implement a chart application and deploy it over the web. Finally, conclusions and future work are given in section 4.

2 Web Objects in XML (WOX)

This section presents the WOX architecture, the operations allowed from a client application, the web browser interface to invoke operations on web objects, and the format of the XML messages exchanged between a WOX client and a WOX server. WOX is a working prototype that can be downloaded from http://algoval.essex.ac.uk/wox/Downloads.html. It is straightforward to install and use. It is also accompanied of a set of client example programs.

2.1 WOX Design

We based the design of WOX on standard object-oriented concepts, distributed object programming and resource-oriented (REST) web services.

The WOX architecture is based on interfaces, which separates two important notions in distributed systems: the definition of behaviour given by the interface, and the implementation of that behaviour. A WOX server will have the implementation of the service, and a WOX client will have only an interface of that service in order to create objects, access them, and execute operations.

The nature of WOX required a strong object-oriented language, which fulfilled all of our requirements. We decided to implement it in Java [4] because it is a platform independent language (although we are also implementing a WOX serializer, and WOX client libraries in C#), which makes our system runnable on a variety of platforms; it is free and not restricted to the use of any commercial tool; and it highlights the concept of an interface, which makes it ideal for a distributed system.

It is important to note that RMI [6] also uses interfaces to allow access to remote objects, but in a very different way, which can be somewhat tedious in practice. We provide in our case study an explanation of the set of steps and changes required in order to implement the chart application using RMI. In this respect, a WOX client only needs a simple interface of the service.

A WOX client makes method invocations on a proxy that is created dynamically for the interface. A dynamic proxy is basically a class that implements a list of interfaces provided at runtime, such that method invocations through one of the interfaces on an instance of the class will be dispatched to another object [16]. The proxy translates the method invocation to XML and sends it to the WOX server. This mechanism uses the proxy design pattern [1].

One of the main ideas behind WOX is to expose remote objects through their own URI [8] and allow clients to have access to them. URIs are used to identify objects inspired by the principles of the REST architectural style. The notion of URI has been widely and successfully used by the web, and it is also a standard way to identify resources. Proponents of REST [10,11,12] argue that objects should be identified through their own URI, because this is how the web works.

Remote references in WOX are URIs, so that a client can refer to a remote object by specifying its URI. The XML encoding of a WOX object can be viewed by typing the URI into the address bar of a standard XML-aware web browser.

The concept of remote reference in WOX is widely used because a client can request a remote reference to a specific object, pass remote references as parameters, and also receive results from method invocations as remote references. A basic example of the mechanism used by WOX in a method invocation is shown in Figure 1, and the detailed steps carried out are described in the following list.

1. The WOX client program invokes a method on a remote reference (the way in which the client invokes a method on a remote reference is exactly the same as that on a local object, as far as the client program is concerned).
2. The WOX dynamic proxy takes the request, serializes it to XML, and sends it through the network to the WOX server.
3. The WOX server takes the request and de-serializes it to a WOX object.
4. The WOX server loads the object and executes the method on it.
5. The result of the method invocation is returned to the WOX server.
6. The WOX server serializes the result to XML and either the real result or a reference to it is sent back to the client. The result is saved in the server in case a reference is sent back.
7. The WOX dynamic proxy receives the result and de-serializes it to the appropriate object (real object or remote reference).
8. The WOX dynamic proxy returns the result to the WOX client program.

From the WOX client program's point of view it just makes the method invocation and gets the result back in a transparent way. WOX can refer to remote objects that reside in the same WOX server as that of the object (a relative URI can be used), or to remote objects located anywhere on the Web.

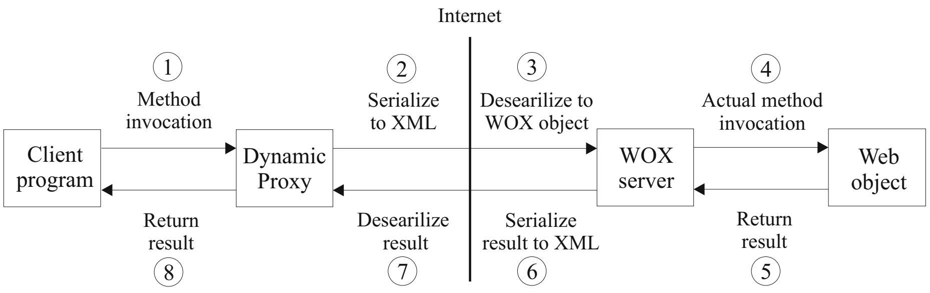


Fig. 1. A remote method invocation in WOX

2.2 WOX Client Operations

This subsection introduces the fundamental operations supported in WOX, and those not covered here are described in [14]. A set of client code examples can be found at `http://algoval.essex.ac.uk/wox/Examples.html`.

Creation of a new object. When a WOX client program requests the creation of an object, the WOX server creates the object and stores it. The WOX server returns to the client, based on the policy chosen, either the actual object or a remote reference to it. The remote reference is specified by a URI that points to the actual object. Once the WOX client program holds the actual object or the remote reference, it can invoke instance methods on it.

```
Chart chart = (Chart) WOXProxy.newObject
                      (serverURL, className, args, policy);
```

The code above creates a `chart` object of type `Chart`, which is a user-defined interface to hold either a real object or a remote reference to it. `WOXProxy` is a class provided by the WOX client libraries to allow the execution of WOX operations like `newObject` that takes four parameters: `serverUrl`, which represents the URL where the WOX server can be contacted to create the object (e.g. `http://csres109:8080/WOXServer/WOXServer.jsp`); `className` is the package qualified class name of the object to be created; `args` are the set of arguments used to construct the object; and `policy` is an integer number that represents whether a real object or a remote reference must be returned. The policy parameter also specifies the default mode in which WOX operates for future method invocations on the object (whether it returns real objects or remote references).

Request for a remote reference. A remote reference is requested by a WOX client program to invoke instance methods on the object to which the remote reference refers to, or simply to use it as a parameter in another method invocation. In this type of request, the WOX server looks for the specific object and returns a reference to it. The code example below gets a remote reference to a `Chart` object. The `objectUrl` parameter is the URL where the object is located. A `WOXException` will be thrown if there is no object at the URL specified.

```
Chart chart = (Chart) WOXProxy.getReference(objectUrl);
```

Static Method Invocation. A static method invocation is similar to a web service remote procedure call (like in SOAP[5], XML-RPC[7], or JSON-RPC[19]), in the sense that the WOX client requires no access to a particular object stored in the WOX server. When a static method invocation is requested, the WOX client invokes the `invokeService` method of `WOXProxy` class, in which it is specified the particular method to be executed (`methodName`), the class to which it belongs (`className`), the set of parameters received by the method (`args`), and the mode of operation (`policy`). The WOX server executes the method and returns the result. A `WOXException` will be thrown if the WOX server cannot find the method with the signature specified.

An important difference between web service remote procedure calls and WOX static method invocations is that a WOX client can also specify using the `policy` parameter, whether the result returned is the actual result, or a reference to it. Moreover, SOAP does not handle remote references, XML-RPC has a limited set of data types, and JSON-RPC does not support marshalling of objects with circular references.

An example of a WOX static method invocation is shown below, in which its return type is the user-defined interface `Manager`. The interface `Manager` is used in this example, but it could be used any other interface or concrete class, as long as it is consistent with the return type of the method invocation and the policy chosen in the WOX operation.

```
Manager manager = (Manager) WOXProxy.invokeService (serverUrl,
                            className, methodName, args, policy);
```

Instance Method Invocation. When a WOX client holds a remote reference, it can invoke instance methods on it in the same way as if it was a local object. The WOX mechanism will redirect the call through the network to the WOX server, which will in turn execute the method on the specific object. Once the method has been executed, the WOX server returns the result to the client. It should be noticed that a remote reference is actually a dynamic proxy on which the WOX client invokes methods. In the code below the client gets a remote reference to a `Chart` object, and then invokes its `getImage` instance method.

```
Chart chart = (Chart) WOXProxy.getReference(objectUrl);
byte[] graph = chart.getImage();
```

Destruction of an object. When a client does not require a remote object any more, it should be destroyed and garbage collected. Our current prototype of WOX includes an operation to explicitly destroy an object by providing its URI. However, in a future release, we could also allow clients to specify how long the object's life should be, and when the time for the destruction was reached then the object would be destroyed (i.e. removed from persistent storage). We appreciate that one of the reasons that SOAP does not allow object references is to eradicate any problems with distributed garbage collection, but believe that for many problem domains, object references are essential. We predict that for particular application areas and user communities, sensible usage policies will evolve given a suitably flexible framework.

2.3 Web Browser Interface

This subsection describes another way of executing WOX operations. As part of its REST foundations, WOX objects can be inspected by an XML-aware web browser. Furthermore, methods can be invoked on persistent objects using a web browser interface. This mode of operation allows clients to execute methods on a specific object without needing a Java client program.

Inspecting a web object. Every object that is persisted after a WOX call is stored in XML and can be inspected by a web browser. Below is the XML representation of a Manager object, which was used in one of our previous examples.

```
<object type="company.Manager" id="0">
  <field name="name">
    <object type="String" id="1">Robin Dyson</object> </field>
  <field name="age" type="int" value="35" />
  <field name="department">
    <object type="String" id="3">Finance</object> </field>
</object>
```

Invoking methods on a web object. Since every object can be identified uniquely, it is also possible to invoke methods on persistent objects through a web browser. A method invocation on a Manager object is as follows.

```
http://csres109:8080/WOXServer4/invokeMethod.jsp?
       objectId=Manager645313585&method=getName
```

The method invocation shown above would invoke the `getName` method of the `Manager645313585` object. The result of the method invocation would be returned as XML, which is the default mode of operation for WOX answers, but it can also be returned as html (by specifying `mode=html` in the query string), in which case only the string would be returned. There is a special case in which WOX can also return an image (`mode=image`) when the return type of a method is an array of bytes. This mode of operation will be presented in the case study.

Using a web browser, WOX is also capable of invoking methods with parameters of primitive data types, but not with parameters of other data types, like user-defined classes. This way of operation through the browser is similar to the way in which Apache Axis [13] allows to invoke methods of classes. The main differences are that Axis, which is SOAP based, does not have the concept of an object, thus the methods are invoked as if they were static methods. Another important difference in this mode of operation is that Axis does not support package qualified classes. In this respect, WOX enables the invocation of methods on any web object, and methods of any package qualified class.

Alternatively, WOX provides a user interface with all the possible methods to invoke on a specific web object. This can be accessed by typing the same URL presented before, but omitting the method parameter. WOX will present all the methods available for invocation on that object. Figure 5 in subsection 3.4 shows this web user interface with some of the methods available for a `Chart` object.

When clicking the *Invoke Method* button of the desired method, a query string is built with all the information needed for the method invocation. The request is made to the WOX server, which will send an answer via an XML message with the result of the method invocation.

2.4 XML Messages in WOX

An XML message in WOX is a request from a WOX client or the response sent back from a WOX server. A request can be any of the operations described in the previous section, while the response could be a real object, a remote reference, or an exception generated by the WOX server.

Since our system is based on object-oriented programming, the XML messages are generated by serializing objects of different classes according to the request made by a client, or the result or exception generated by the server. Some of these classes are shown in Figure 2, which contain attributes with all the information required to accomplish the request made. For example, the `WOXConstructor` class represents a request of object creation, where the `className` attribute specifies the name of the class of the object to be created, `types` and `args` are arrays that contain the parameters needed to construct an instance of the class,

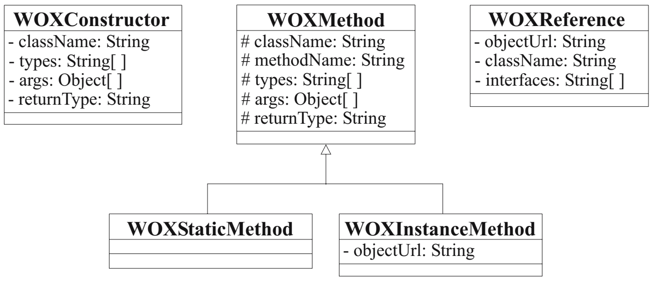


Fig. 2. Some WOX classes

and `returnType` indicates whether a remote reference or a real object must be returned. The `WOXMethod`, `WOXStaticMethod` and `WOXInstanceMethod` classes represent method invocations; and `WOXReference` remote references.

In addition to the classes shown in Figure 2, there are some other classes to represent all of the operations described in the previous section, such as `WOXDestructor`, `WOXUpdate`, `WOXUpload`, etc. Similarly there are classes to represent exceptions thrown by a WOX server, which inherit from `WOXException`.

The XML message shown below represents a static method invocation (an object of the `WOXStaticMethod` class). Although our WOX prototype (WOX serializer, WOX client libraries, and WOX server) is only implemented in Java (we are implementing a WOX serializer, and part of our WOX client libraries in the C# programming language), the XML messages should be appropriate for any other object-oriented programming language.

```
<object type="server.WOXStaticMethod" id="0">
  <field name="className">
    <object type="String" id="1">problems.test.MathClass</object>
  </field>
  <field name="methodName">
    <object type="String" id="2">returnArrayInt</object> </field>
  <field name="types">
    <array type="String" length="1" id="3">
      <object type="String" id="4">int</object> </array> </field>
  <field name="args">
    <array type="Object" length="1" id="5">
      <object type="Integer" id="6">5</object> </array> </field>
  <field name="returnType">
    <object type="String" id="7">Copy</object> </field>
</object>
```

2.5 WOX Server Operation

Every request of a WOX client is received by the WOX server as an XML message, it is de-serialized to a `WOXAction` object, and its `doAction` method is

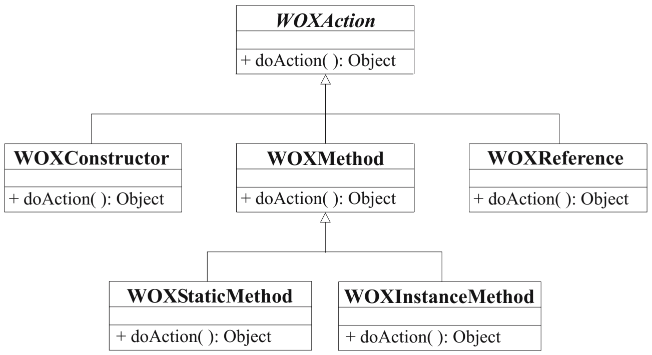


Fig. 3. `WOXAction` hierarchy diagram

executed. Figure 3 illustrates the hierarchy diagram for the `WOXAction` class. `WOXConstructor`, `WOXInstanceMethod`, `WOXStaticMethod` and `WOXReference` extend `WOXAction`. This mechanism allows a `WOXAction` object to invoke the `doAction` method of the appropriate class, which is coded differently, based on the type of request. This design is very flexible in the sense that there can be any other types of new requests without modifying the existing code. New types of requests would be represented as classes that extend the `WOXAction` class.

2.6 Limitations

WOX in its current state has also limitations, that somewhat can be seen as features not included in this release. A list is presented along with an explanation:

- Language independence: Although all the messages are represented in XML, WOX server and WOX client libraries have been only developed using the Java programming language, but we believe that the XML messages generated by WOX are appropriate to be implemented in any other class-based object-oriented programming language, such as C#, C++, Ruby, or Smalltalk. Our initial approach has been to develop our WOX serializer in C#.

The main issue to be considered when implementing WOX in other object-oriented programming languages is the serialization process, which is actually how objects will be represented in XML. This leads to consider multiple inheritance, and other programming language-specific features, which would be represented in the XML message.

- Security and ownership of objects: WOX does not support the concept of ownership of an object nor security policies for accessing web objects. Any client can have access to any objects created previously by another client. There is no restriction on who is executing what method of what object. It is just necessary for a client to have the URL to be able to access the object. For most practical applications, this would be a severe limitation, but at the prototype stage we did not want to be distracted by these considerations. However, since WOX is layered over HTTP, any HTTP-based security mechanism could be used.

- Asynchronous processes: All the operations in WOX are synchronous, which means that a result is immediately sent back to the caller. Asynchronous operations would allow clients to submit their jobs or processes and wait for results. Results would be returned to the caller normally via a callback operation.

- Object navigation: All the objects that persist after a WOX call can be inspected through an XML-aware web browser. An additional feature in WOX would allow clients to navigate through object graphs and be able to request specific nodes (objects). The use of an object-oriented db as the persistent storage for objects, such as db4o[17], could facilitate the implementation of this feature.

3 Case Study

This section presents a case study, in which a chart application is exposed through the Internet using three different technologies: RMI as a distributed object technology; SOAP as a web service technology; and finally our WOX prototype as a technology with features from both paradigms. The next subsections describe the chart application, and focus on the set of steps needed to implement it in the three different technologies. A different case study, describing a pattern recognition application, can be found in [15]. The case study presented in this section shows how best to deploy this chart application over the web with existing technologies and WOX. We chose this simple case study to introduce WOX, but we are already working on a more realistic application (the development of a game server) to demonstrate our ongoing work, which covers some of the limitations described in the subsection 2.6.

3.1 The Chart Application

The chart application we want to expose through the Web is used to input the data we collect from our experiments, get a statistical summary of the data, and get an image with a line graph of the data provided. The way in which the chart application works is very simple. We generate a new `Chart` object for every experiment, which will collect all the data for that particular experiment. The following line of code would create a `LineChart` object labeled *WOX Experiment*.

```
Chart chart = new LineChart("WOX Experiment");
```

Every time a new result from an experiment is ready it can be added to the `chart` object. The following code gets a new result from an experiment and adds it to the previously created `Chart` object. Since this is only for demonstration purposes we are using the `getResult` static method of the `Experiment` class (which returns a randomly generated `double` value), but this could be easily a method of any object which actually returns a result.

```
double x = Experiment.getResult();
chart.addValue(x);
```

Values are added to a `chart` object as results come from an experiment. Once the experiments have finished we can invoke the `getImage` method to get an array of bytes representing a line chart image.

```
byte[] image = chart.getImage();
```

In this simple application, the `chart` object would be created once, and experiments could be run over the world and use the `chart` object to add new results to it. This would also allow you to get a graph with the results at any point in time. This application is clearly state-based, because it needs to do the data collection, which will serve to do the statistical summary and generate the graph. The application consists of the `Chart` interface and the `LineChart` class.

The aim of the case study is to evaluate how well the deployment of the chart application is supported in the three different technologies: RMI, SOAP, and WOX. The deployment of this application through the Internet will allow clients ideally to refer remotely to chart objects that were created previously. In that way, clients do not need to hold `chart` data objects in their own computers, and they can also eventually save time by requesting a reference to an existing `chart` object. They can also add new values to the `chart` object and request an updated graph. A more realistic example would also allow chart styles to be set up and referenced. The implementation of this application will need to maintain the state of the objects and some mechanism to handle remote references.

3.2 Implementation Using RMI

In order to implement the chart application in RMI, several changes must be made to the original java source files, and follow a list of steps to make the objects remotely accessible to client applications. Since this is a very simple application the changes required will be only to the `Chart` interface and the `LineChart` class.

Modifying the original classes. The set of modifications to the classes in the chart application are described in the following list.

- Select the interface that clients will be using to access remote objects on the server. In this case the `Chart` interface would be used for this purpose.
- The `Chart` interface must extend the `java.rmi.Remote` interface provided in the Java API. The `java.rmi.RemoteException` exception must be thrown by all its method signatures.
- The `LineChart` class also needs to throw the `java.rmi.RemoteException` exception in every one of its methods and constructors.
- Those classes that will be traveling through the network must implement the `java.io.Serializable` interface. This is the case for the `LineChart` class.
- In order to expose `chart` objects remotely it is necessary either to extend the `java.rmi.server.UnicastRemoteObject` class or to specify that in the constructor of the class, when the object is created. In this case `LineChart` class must extend the `UnicastRemoteObject` class.

An alternative solution to implement this application using RMI could be to provide wrapper classes for every class or interface that requires modifications.

Those new wrapper classes would contain the modifications described in this section, and the original source classes would not be affected.

Deploying and running the application. Once the source files have been changed as described,the following steps must be carried out:

- Generate the stub for the remote interface `Chart`, by executing the `rmic` stub compiler (even though this is no longer required in Java 5 or later version, as dynamic proxies are generated). A client application will need the remote interface and the stub generated in order to access the remote objects.

- On the server side computer it will be required to copy all the classes and interfaces modified and start up the *Object Registry* (this is where client applications will find the remote objects). A server program needs to be written to create and register some objects in the *Object Registry*. In this case, the server program will have to create `LineChart` objects. These objects will actually be the remote objects available to the clients. It must be noticed that this server program creates some objects, which will be available for clients to access. Some extra methods would have to be provided in the `LineChart` class in order to allow clients to create their own objects.

- Client applications require the interface `Chart`, in addition to the stub generated by the `rmic` compiler (stubs are not required for Java 5). A sample code for a client application that uses RMI is shown below. The code gets a remote reference to a `chart` object, which has been created by the server program in the *Object Registry*. The client application can work with the remote reference as if it was a local object. It adds a new value to the `chart` object and then it gets an updated image with the line graph. One restriction as stated before is that clients are not able to create their own remote `chart` objects directly, even though extra methods could be provided to do so.

```
Chart chart = (Chart) Naming.lookup("chart01");
double x = Experiment.getResult();
chart.addValue(x);
byte[] image = chart.getImage();
```

Extra work would be needed to expose those objects over the Web, in which RMI tunnels its requests through HTTP. We could also have chosen to implement the entire chart application with Enterprise JavaBeans (EJB) Technology[20] (which is a more powerful technology that communicates through RMI) and an application server, such as JBOSS[21]; but we know that EJBs introduce many more unnecessary steps for this simple type of application. We would have had to deal additionally with EJB and home objects, home and local interfaces, and deployment descriptors.

3.3 Implementation Using SOAP

The implementation of the chart application using SOAP needs many more changes than those in RMI, because the application requires maintenance of the state of the objects between method calls, which SOAP does not support. In order for SOAP to refer to remote objects a considerable extra programming effort is required, which leads to changes in the source classes or the creation of wrapper classes to encapsulate the instantiation and maintenance of remote objects. We prefer the latter method.

Creating wrapper classes. Following the idea of writing wrapper classes, there must be a mechanism to maintain the objects created by the client, and to refer to them. Figure 4 illustrates a class diagram that shows how to wrap up the original classes in the chart application in order to be able to expose chart objects remotely. The following changes are needed:

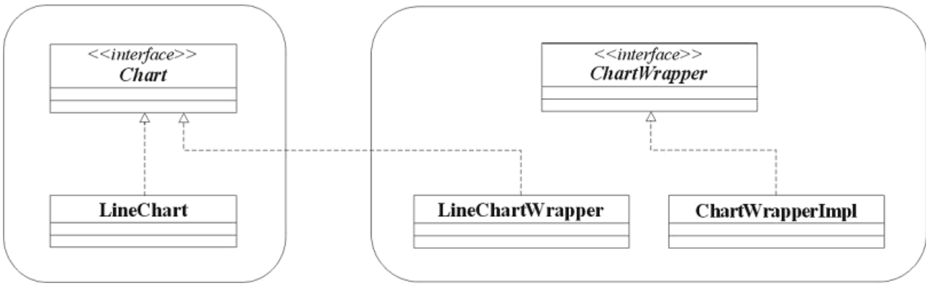


Fig. 4. Wrapper classes for the chart application

- A `ChartWrapper` interface that represents the SOAP interface of the service, which wraps up the `Chart` interface. `ChartWrapper` takes each instance method of the original `Chart` interface, and adds an id parameter to them. An example is illustrated below, where the signature of the `addValue` method in the `ChartWrapper` interface has now an additional parameter.

```
public void addValue(String id, double x);
```

- A `ChartWrapperImpl` class which is an implementation of `ChartWrapper` interface, and maintains a map of chart objects. The map can be maintained in memory or persistent storage. This implementation of the service will allow maintaining the state of chart objects on the server.

- A `LineChartWrapper` class that implements the `Chart` interface and provides a wrapper for `LineChart` class on the client side. It will be the interface of the service to the client. Clients will create chart objects of `LineChartWrapper` class instead of `LineChart` class, as can be seen from the code below. The class `LineChartWrapper` contains the logic to maintain the state of a chart object on the client side by keeping track of the id sent by the server to identify a specific chart object. `LineChartWrapper` class uses a proxy mechanism to send the requests from the client to the appropriate web service on the server.

```
Chart chart = new LineChartWrapper("WOX Experiment");
double x = Experiment.getResult();
```

```
chart.addValue(x);
byte[] image = chart.getImage();
```

- A Proxy class that receives the request from the client, extracts from it the web service name to be executed and the set of input parameters needed to invoke it. Each parameter must be mapped to the appropriate XML data type by using the serializers provided in SOAP, which can only serialize primitive data types, arrays, vectors, and user-defined classes that follow the Java Bean conventions. If there are other classes in the application to be serialized, then it would be also required to provide serializers for them. Those provided in SOAP require the classes to be modified to follow certain conventions in order to work properly. For example if a bean serializer is used, then the class to be serialized must follow the Java Bean conventions. On the other hand, one can write its own custom serializers, though it is a demanding and time-consuming task.

- The classes in the original chart application were modified to provide set and get methods for all their attributes, in order for them to be serialized properly.

Deploying and running the application. The deployment of the SOAP-version of this chart application involves the following steps.

- Copy to the server all those classes of the original application, in addition to `ChartWrapper` and `ChartWrapperImpl`, which define the SOAP service.

- Write a deployment descriptor to actually deploy the chart service on the server. The deployment descriptor specifies the name of the web service, the java class to be used for the service, the methods that can be invoked by clients, the scope of the web service, and the type mappings, which define the serializers to be used for user-defined classes.

- On the client side it will be required the classes of the original application, in addition to the Proxy class and the `LineChartWrapper` class, which will be the interface of the service to the clients.

- Running the application involves creating a `chart` object, adding some values to it, and getting the image with the line chart.

Despite the application is functional after all the modifications made and the lengthy procedure, it still has some drawbacks in the serialization efficiency. SOAP lacks of an efficient way for serializing arrays of primitive datatypes. Table 1 in subsection 3.4 shows a comparison in time and storage space between WOX and SOAP when they serialize an array of a primitive data type.

3.4 Implementation Using WOX

The implementation of this chart application in WOX is straightforward. For this application there is no need to create stubs for clients, rewrite classes to extend or implement interfaces, change method signatures to throw remote exceptions, write wrapper classes, or any extra programming effort, as long as the services to be exposed have an interface and an implementation.

The `Chart` interface and the `LineChart` class will reside on the server side, with no modifications. The client application will need the `Chart` interface in order to create new remote chart objects, access them, and invoke methods on them. Note that there could be more Chart implementations added to the server, and there would be no need to edit any configuration files, or recompile any classes. The only requirement is that the WOX server can locate the necessary classes to execute the implemented methods. However if methods were added to the interface, the client would need the new interface in order to invoke the new methods. A WOX client would use the fragment of code shown below to use the chart application.

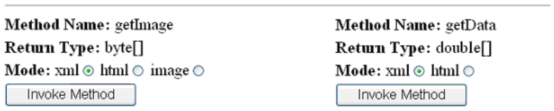
```
Chart chart = (Chart)WOXProxy.newObject(serverUrl,classN,args,pol);
double x = Experiment.getResult();
chart.addValue(x);
byte[] image = chart.getImage();
```

The first statement creates a Chart object. The serverUrl is the URL of the WOX server; classN is the class of the object to be created (stats.LineChart in this case); args are the set of arguments used to construct the object; and pol is an integer value that represents whether the real object or a reference to it must be returned from the WOX server (we specify that a remote reference must be returned, since we want the chart objects on the server side).

Requesting a remote reference of a chart object is particularly useful in this application that needs to preserve the state of the object, which is modified by adding values to the chart. In cases such as this the ability of WOX to allow clients to create and manipulate objects on the server becomes essential. While it is possible to do the same with SOAP with considerable extra programming effort, as we have shown in the previous subsection, the difference is that WOX actively supports this stateful style of interaction.

The process of creating a remote object continues when the WOX server receives the request, creates the chart object and returns the remote reference to the client. When the WOX client receives the remote reference to the new object, it creates a proxy that implements the Chart interface. This proxy will be used to make the subsequent method invocations on the remote object. The third and fourth code statements from the code above are adding a new value to the chart and getting an image with a line graph.

Since adding values to a chart object and getting an image with a graph are method invocations on a chart object, they can also be executed through the web browser user interface that WOX provides. Figure 5 shows the user interface provided by WOX to invoke methods on the chart object previously created. The getImage method has 3 different modes of operation: xml to return the array of byte serialized in xml, html to get the plain array of byte, and image to get the actual image shown in the web browser. This mode of interaction is only possible in the web browser interface. It actually uses the *image/jpeg MIME* type [18] to decode the array of byte when it is sent to the web browser. The possibility to include other MIME types to a WOX server would allow to decode array of bytes into specific formats that can be displayed by a web browser (e.g. audio and image files, text documents, etc.).


Method Name: getImage
Return Type: byte[]
Mode: xml [✓ ●] html ○ image ○
Invoke Method

Method Name: getData
Return Type: double[]
Mode: xml [✓ ●] html ○
Invoke Method

Fig. 5. Web browser interface to invoke methods on a `Chart` object

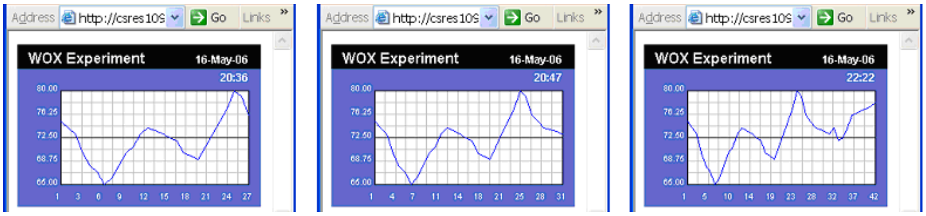

Address http://csres109 Go Links
WOX Experiment 16-May-06 20:36
WOX Experiment 16-May-06 20:47
WOX Experiment 16-May-06 22:22

Fig. 6. Method invocations through the web browser

It can also be possible to build a URL with a query string specifying the method to be invoked, the parameters needed, and the mode of operation. Figure 6 shows three different invocations of the `getImage` method using the `image` mode. Clients can access the `chart` object either via a Java client program, through this web browser interface, or simply by building a URL with an appropriate query string.

A fragment of the XML message returned from the WOX server with the array of `byte` is shown below. It is encoded using base64.

```
<array type="byte" length="3023" id="0">
    iVBORw0KGgoAAAANSUhEUgAAAQ4AAAC0CAIAAADq9VVVAAAIg0lEQVR42
    <!-- rest of array omitted -->
</array>
```

WOX serialization process transfers data in XML in a more optimized way than SOAP. By default, SOAP uses an XML element for each element of an array of primitive elements (such as int for example). This means that SOAP-encoded arrays can be over 40 times the size of their binary encoding. The exception to this are `byte` arrays like the one illustrated above, which are encoded efficiently using base-64 (which WOX also uses for byte arrays). Given the speed of modern computers, and the fact that many of us have access to high bandwidth Internet connections, this difference in encoding efficiency might seem unimportant. However, Table 1 emphasizes how significant this difference is, both in time and space usage. For arrays of more than 30,000 int, the SOAP server (Apache Axis) crashed with an out of memory error.

Using WOX to implement the chart application allows other clients to have access to chart objects remotely through their own URI, either by using a Java client program or the web browser user interface. Client applications are also able to create their own `chart` objects, and add new values to existing ones.

Table 1. Time and space usage for passing an array of 20,000 int in WOX and SOAP

Method	Time (ms)	Size (KB)
WOX	80	106
SOAP	3,300	4,200

4 Conclusions and Future Work

In this paper we introduced WOX (Web Objects in XML), which is a web distributed object protocol that allows remote method invocations on web objects, and remote procedure calls on exposed web services. WOX uses HTTP as its transport protocol and XML to encode the messages exchanged between client and server. WOX exposes object references as URIs, inspired by the principles of the Representational State Transfer architectural style. Using URIs in this way allows parameters to be passed, and values returned, either by value or by reference. WOX objects can also be accessed through a web browser interface, from which methods invocations can be executed. We described the WOX architecture, the set of client operations supported, the web browser interface, the format of the XML messages, and the WOX operation modes.

We also presented a case study, in which a state-based chart application is described and exposed over the Internet using three different technologies: RMI, SOAP and WOX. WOX proves to be the most straightforward system for implementing this type of application. Applications like the one presented in this paper, which has some special features such as being accessible remotely over the Internet, maintaining the state of objects, having access to remote objects, storing them in a standard text format (XML), among others, can be implemented using WOX. The possibility to inspect the object through an XML-aware web browser and execute method invocations on web objects via a web browser are also built-in features of WOX. The ease of use to implement this type of applications is another of its advantages over the other technologies discussed.

The limitations or features not included in WOX have also been discussed. They include the language independence given by the XML messages generated by WOX; the security and ownership of objects; the support for asynchronous processes; and the possibility to navigate through web objects. Even in its current state, however, we are already putting WOX to good use, and find it to be a simple, easy to use, and robust protocol.

References

1. Gamma, E., Halm, R., Johnson, R., Vlissides, J.: Design Patterns: elements of reusable object-oriented software. Addison-Wesley, Reading (1995)
2. Common Object Request Broker Architecture (CORBA) Object Management Group (2000), available at `http://www.omg.org`

3. Extensible Markup Language (XML), World Wide Web Consortium, available at http://www.w3.org/TR/REC-xml/
4. Java Technology Sun Microsystems (1994), available at http://java.sun.com
5. Latest SOAP Versions, World Wide Web Consortium (2003), available at http://www.w3.org/TR/soap/
6. Wollrath, A., Waldo, J.: The Java Tutorial, Trail: RMI Sun Microsystems, available at http://java.sun.com/docs/books/tutorial/rmi/
7. Winer, D.: XML-RPC Specification, available at http://www.xmlrpc.com/spec
8. Berners-Lee, T.: Universal Resource Identifiers - Axioms of Web Architecture, World Wide Web Consortium, available at http://www.w3.org/DesignIssues/Axioms.html
9. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures, available at http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
10. Costello, R.: Building Web Services the REST Way, xFront, available at http://www.xfront.com/REST-Web-Services.html
11. Prescod, P.: Second Generation of Web Services, available at http://webservices.xml.com/pub/a/ws/2002/02/06/rest.html
12. He, H.: Implementing REST Web Services: Best Practices and Guidelines, available at http://www.xml.com/pub/a/2004/08/11/rest.html
13. Web Services - Axis, available at http://ws.apache.org/axis/ ASF, 2004
14. Jaimez González, C., Lucas, S.: Web Objects in XML: a Web Protocol for Distributed Objects, Technical Report, University of Essex (2005)
15. Jaimez González, C., Lucas, S.: Implementing a Pattern Recognition Application Using RMI, SOAP and WOX, Technical Report, University of Essex (2005)
16. Dynamic proxy classes, Sun Microsystems (1999), available at http://java.sun.com/j2se/1.3/docs/guide/reflection/proxy.html
17. db4o database (2006), available at http://www.db4objects.com/,db4objects
18. MIME Media Types, Internet Assigned Numbers Authority (1999), available at http://www.iana.org/assignments/media-types/
19. JSON-RPC 1.1 Specification Working Draft (2006), available at http://json-rpc.org/wd/JSON-RPC-1-1-WD-20060807.html
20. Enterprise JavaBeans Technology, Java Platform, Enterprise Edition (Java EE) (2007), available at http://java.sun.com/products/ejb/
21. JBoss Application Server (2007), http://www.jboss.org/products/jbossas

Experience with Dynamic Crosscutting in Cougaar

John Zinky, Richard Shapiro, Sarah Siracuse, and Todd Wright

BBN Technology
Cambridge, MA USA
{JZinky, RShapiro, SSiracuse, TWright}@bbn.com
http://Cougaar.org, http://quo.bbn.com

Abstract. Component-based middleware frameworks that support distributed agent societies have proven to be very useful in a variety of domains. Such frameworks must include support for both agents to implement business logic and runtime adaptation to overcome the inherent limitations of unreliable, resource-constrained environments. Regardless of how any particular middleware framework is organized into components, the business logic and adaptation support will inevitably require some crosscutting of the dominant decomposition. In this paper, we discuss a spectrum of dynamic crosscutting techniques in support of runtime adaptation that we have implemented in Cougaar, a component-based service-oriented architecture. We describe these crosscutting techniques and show how they can be used to enhance the flexibility and survivability of agent-based applications.

Keywords: Aspect-Oriented Programming, Component-Based Middleware, Distributed Agents.

1 Introduction

Writing distributed applications would be simple if programmers only had to implement the input and output behaviors of the domain functionality (or "business logic"). However, the majority of the code written for real-world distributed applications is in service of systemic issues, such as security, reliability, performance, and deployability, and not domain issues. Ideally, a distributed applications framework should hide the systemic issues altogether. This ideal isn't currently feasible, but a next-best alternative is separating the systemic code from the application code so that each can be developed and tested independently. The systemic support would be the responsibility of the framework, leaving the application developer free to concentrate on the business logic.

The key difficulty in creating such a framework is that the systemic concerns crosscut the application's domain-specific decomposition. In particular, systemic code can't be written as fully self-contained components, but must instead be inserted at multiple places in domain components. Further, several independent system concerns might need to insert code at the same place in a given domain component. For example, an access-control concern needs to add a check to each of an object's methods to determine if the caller has the privilege to execute that method, while a performance concern might want to limit the rate of method execution so that underlying resources will not be over-utilized. Both of these concerns need to attach systemic behavior at the same points in the domain code, in this case before domain method calls.

Many available techniques enable middleware frameworks to support both systemic concerns and application concerns by opening up the application execution in order to allow system code to run. Two important distinctions among these approaches are, first, when in the application life-cycle this code insertion occurs, and, second, how much burden the techniques put on the application programmer and the runtime environment. In this paper we focus on our experience using dynamic techniques for inserting systemic behavior at runtime.

Specifically, we will describe our experiences in making the Cougaar distributed agent framework survivable. Our goal was not to create new Aspect-Oriented Programming (AOP) mechanisms or new software patterns, but rather to use both disciplines together to create techniques for meeting the runtime-adaptability of Cougaar agent societies. We present several small case studies to show the uses of these dynamic AOP mechanisms and how they interact. These examples show non-trivial uses of dynamic AOP techniques in the context of large-scale runtime-adaptive distributed systems.

Cougaar is a Java-based middleware framework for the construction of large-scale distributed agent-based applications [7]. A large logistics planning society was developed using Cougaar, with more than 1000 agents running on more than 100 hosts. To test the feasibility of fielding such a large system in an unreliable and resource-constrained environment, DARPA started the Ultralog project in 2001 to add survivability features to Cougaar. The goal of the Ultralog project [20] was for the distributed application to continue operating with up to 45% information infrastructure loss with minimal loss in functional capabilities or performance degradation.

A component-based framework and most of the dynamic AOP techniques described later in this paper were added to the Cougaar framework to meet this requirement. Among the significant enhancements was a Service-Oriented Architecture (SOA) similar to Java Enterprise Beans [12], but extended to allow crosscutting among infrastructure components and their services. With Cougaar augmented as described below, the goals of Ultralog were met: experimental runs continued to perform their logistics planning function as the agent society reconfigured itself on the remaining infrastructure after 45% of the hosts failed [19].

This paper will address only the underlying Cougaar framework mechanisms that supported the development and execution of the systemic survivability code, rather than the algorithms or the agent societies that implemented them. The survivability mechanisms themselves are addressed elsewhere [19]. More specifically, we will address the use of dynamic crosscutting techniques at the component level to enable survivability. These survivable behaviors adapt at runtime to meet the system requirements of the agent-based logistics application, within the resource constraints of the underlying physical infrastructure. The crosscutting mechanisms described here were used by multiple programming groups, tested in large-scale experiments, and are the basis for the deployment of a distributed logistics application which runs in an unreliable resource-constrained environment [19].

This paper is organized into the following sections. Section 2 describes Cougaar's requirements for dynamic crosscutting using the lens of dynamic AOP. Section 3 describes techniques Cougaar uses to address the dynamic crosscutting requirements. Each technique has power and restrictions on when and where crosscutting advice can be inserted into the Cougaar Framework. Section 4 describes small case studies of using dynamic crosscutting to insert adaptive-behavior into the framework. Section 5 describes the relative overhead of adding crosscutting mechanisms to distributed systems. Section 6 describes how other distributed system frameworks address the dynamic crosscutting requirements. Finally, we conclude with a summary of our experience.

2 Requirements for Dynamic Crosscutting

Typically the core concern of any application framework is the application's business logic, while a secondary concern is systemic behavior. But in some situations, a systemic concern is the core decomposition and the business functionality is woven into the system decomposition. For example, hard real-time systems are primarily concerned with ensuring that all events happen within prescribed deadlines [28]. A real-time framework consists of tasks with known execution times; at runtime these tasks are meticulously scheduled so that no deadline is missed. The application functionality cuts across these tasks: the application's natural structure is obscured in favor of guaranteeing the real-time systemic concern. Regardless of which concern is dominant, other concerns will need to crosscut behavior at multiple locations in the dominant concern's execution. Aspect-Oriented Programming (AOP) [1,2,3,4,5] was created to address these crosscutting issues. We will use AOP terminology to describe the Cougaar crosscutting requirements and techniques.

AOP can be understood as the desire to make quantified statements about the behavior of programs, and have the quantification hold over programs written by programmers who are unaware of this additional behavior [29]. AOP defines the following concepts [30]. A *join point* is a specific well-defined event in the control flow of the executed program. *Advice* is behavior that is triggered by a certain event and that can be inserted into the control flow when a specific join point is reached. A *point cut* is a declaration that tells the aspect composition framework which advice to apply at which join point. So in the context of AOP, crosscutting requirements can be categorized by the kinds of quantifications allowed (join point), the kind of behavior that can be asserted (advice), and the mechanisms for combining the base behavior with the asserted behavior (point cut). Dynamic crosscutting adds another dimension: when the quantifications are calculated.

The systemic concerns of the Cougaar framework include security, robustness, storage management, bandwidth management, load balancing, and interoperability. Many of these concerns were developed in parallel, by different groups, for different runtime environments.

To allow these concerns to be combined at runtime, the Cougaar framework needed to support the following dynamic crosscutting requirements:

Configuration Flexibility: Cougaar needs to run over a wide variety of infrastructures and host a wide variety of applications. The framework should be flexible enough to allow only the necessary services and extensions to be loaded. In AOP terms, different concerns should be instantiated and woven together at load time at the earliest. For example, an application that will only run inside a firewall does not need certain security extensions that would be essential for an application running on the unprotected Internet, and should not be forced to pay the price of those unnecessary extensions.

Exposing Join Points: Framework concerns should not be implemented as black boxes. They should expose join points, i.e., points at which new functionality can be added [9]. The framework should allow new services to be added and old services to be overridden, for example by adding a MIME plug-in to a browser. In addition, complicated services should be decomposed to allow new functionality to be added at key times in the service's life-cycle and usage pattern.

Wrappers: The basic framework services should allow extensions to their implementations. The simplest form of extension is the Decorator pattern [15], in which the original implementation of a service stays intact, but can be wrapped with additional functionality. A service can be viewed as a join point at which the wrapper adds advice. For example, a name lookup service could be wrapped with a cache that remembers the results of previous queries.

Chain of Responsibility: Multiple concerns might want to wrap the same service. The Chain of Responsibility Pattern [13,15] allows a chain of delegates to execute before and after each method call to a service. Multiple concerns should be able to add advice to the same join point. For example, the name lookup service could add delegates to handle caching, authorization, and usage monitoring.

Data-flow interception: Some concerns need access to multiple points in a service's internal data flow. The Interceptor Pattern [14] allows a component to subscribe to multiple events exposed by a service implementation. This implies that a concern has state and processing that can be accessed by advice bound to various join points. For example, the name service could expose internal listener interfaces that are invoked when it sends a message to a remote name server or receives an error message from the remote name server.

Runtime Adaptation: Some concerns are designed to change their behavior based on changes in Quality of Service (QoS) requirements or changes in the constraints on resources. The Cougaar framework should include services for obtaining information about the status of applications and resources, as well as the ability to inject advice dynamically based on an evaluation of this information. For example, a concern might decide whether or not to compress a message based on the bandwidth of the path between the source and destination and the amount of available CPU at the end-points.

3 Spectrum of Crosscutting Techniques

The standard spectrum of Cougaar-based crosscutting techniques are implemented as extensions to Cougaar's component model and Service-Oriented Architecture. Join points are defined in the context of the life-cycle of components and their inter-component services. For example, the techniques described in this section allow advice to be inserted when services are registered, resolved, or invoked. These crosscutting techniques gave the Cougaar framework additional power vis-a-vis standard SOA techniques and allowed us to separate independently developed concerns into components that were composed into an adaptive runtime system.

Additional crosscutting techniques, such as the use of AspectJ [6] and byte code manipulation [22, 31], can be used in Cougaar but are not addressed in this paper. These other techniques were immature at the time when we began adding the runtime adaptation enhancements to Cougaar, so we were restricted to traditional Object-Oriented techniques. Further, we wanted to explore crosscutting techniques within the bounds of our service-oriented component-based architecture. A discussion of other AOP tools can be found in Section 6.

An application built on the Cougaar framework is a collection of components that are composed at load-time. XML files specify which components to load and in what order to load them. The application's business logic is implemented by agent-internal components, called *plugins*, which interact through service APIs. Agents interact with each other through services advertised by the Cougaar runtime environment, which is likewise created out of components. Each systemic concern is also implemented as a component and can offer services to other concerns. This mesh of components connected by services defines the dataflow of the system. Crosscutting techniques allow additional components to add advice to this mesh dynamically, primarily by intercepting service calls. This section briefly describes each of the crosscutting techniques in turn, using AOP terminology.

3.1 Service-Oriented Architecture and Components

Cougaar exposes join points using the life-cycle features of a component model and the service registry and the lookup features of Service-Oriented Architecture (SOA). The Cougaar component model is closely related to Java beans [5], with some novel extensions. The core of the component model is the usual container/component tree structure: each *component* belongs to exactly one *container*, which may contain any number of components. The basic Cougaar process (called a Node) defines a set of *insertion points*, which define the structure of the Cougaar middleware components and Agents. For example, each Agent has an insertion point, as well as each subsystem in the runtime environment (some examples of subsystems are message transport and thread management). A Cougaar application is defined at configuration time as a set of components to load into specific insertion points. Adding or removing components can change the middleware's QoS adaptive behavior, as well as the Agent's business behavior.

Inter-component communication happens via *services*. The SOA implementation of a service is provided by a *service provider*, which registers its services with a *service broker* at any level of the component hierarchy. A container controls which services are offered at its layer; it can propagate inherited services from its parent container, override them, block them entirely, or define local services that are unavailable elsewhere.

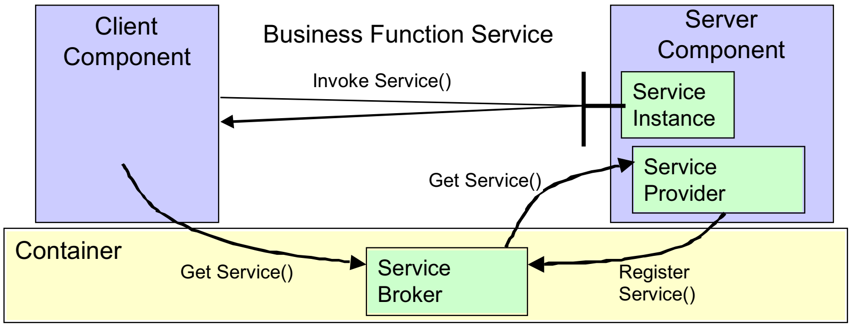


Fig. 1. Cougaar Component Model supports Service-Oriented Architecture

Simple SOA-based adaptation can choose which component will implement a given service. For example, Cougaar has different implementations for whole subsystems, such as the Thread Management Service or the Message Transport Service. At component load time, the choice of implementation is based on the expected environment in which the society will run. In other cases, multiple implementations are loaded and registered, and the decision of which to use is deferred until service lookup time. Finally, a client could bind to both service instances and make the choice at invocation time. Effective adaptation depends on the availability of multiple ways of performing a given service and criteria for choosing the best way in a given situation.

Each interface in the component model is a potential join point. Since the interfaces are not bound until runtime, component instantiation, service registry, service lookup, and service invocation are all opportunities for adding advice. The following techniques allow advice to be added at these join points.

3.2 Binders

Cougaar extends the basic component model by placing intermediaries called *binders* between a container and its components. Components access service brokers indirectly through their binder. Binders can restrict access to services, offer their own services, or modify the services offered by the container. Binders are added when the component is attached to its container. Figure 2 shows that binders can add proxies to both the client and the server sides of a service invocation. Server-side binders implement the Decorator pattern directly, by creating a composed object that implements the behavior for both the component and the binder. A binder offers a form of service delegation to the component it binds. The Service Broker provided by a binder can transparently replace a service reference with a proxy that will delegate to that reference only under certain circumstances. Such proxies also provide a natural point at which systemic information can be gathered and systemic control effected.

A binder can dynamically add advice to the join points associated with a service interface. When the service is resolved the binder can decide whether or not to add the service proxy, based on the binder's point cut specification. If the proxy is added, it can make further restriction on when to run its advice at invocation time. For example, binders can be used to implement security concerns either by restricting the

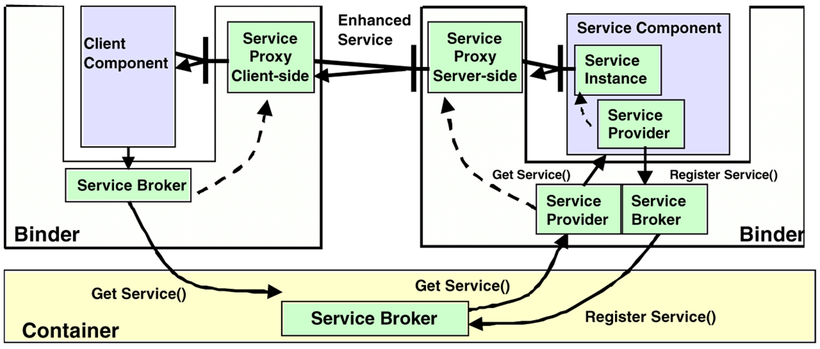


Fig. 2. Cougaar Binder supports the wrapping of a component with additional behavior

component's access to external services (since all service lookups by the component must go through the service broker supplied by the binder) or by restricting access to services the component itself provides (since service registrations made by the component must likewise go through the service broker supplied by the binder).

3.3 Nested Binders

If several concerns need to add advice to a given component, the combined behavior of the component and the concerns must be composed into a new component. The Chain of Responsibility pattern accomplishes this by placing multiple delegates between the client and server components. Binders offer a convenient mechanism for constructing such a chain of delegates, since they can be stacked to insert multiple concerns. A binder can be added to a container to form a systemically controlled environment in which to instantiate a component. Likewise, a binder can be put around a component to create a wrapped component. The stack of binders can be arbitrarily deep, composing multiple concerns.

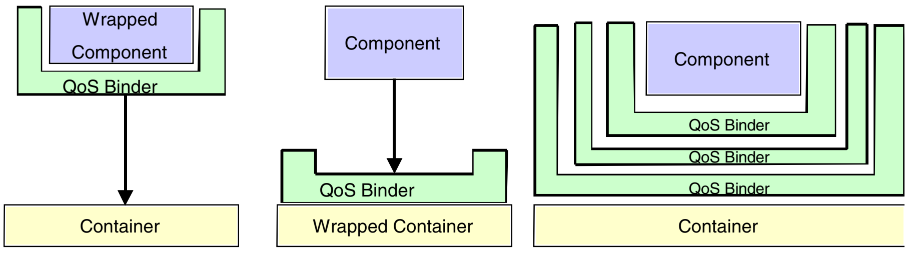


Fig. 3. Nested Cougaar Binders support wrapping a component with multiple behaviors

Cougaar uses nested binders to enforce configurable and dynamic security policies. Binders were created for multiple security concerns, such as access control, denial of service protection, and security logging. For example, if a read-only client attempts to resolve a security-wrapped service, a proxy could be added to throw exceptions if the client tries to access the service's write methods. No proxy would be added for other clients, which has the side benefit of avoiding any per-invocation overhead for the security concern.

3.4 Aspect Interceptors

Some concerns need to coordinate advice inserted at multiple join points. The interceptor pattern allows a subsystem to expose multiple interfaces to add advice to its internal data-flow. The Aspect Interceptor Pattern [10, 36] extends the interceptor pattern to explicitly allow the crosscutting of multiple interfaces. Aspects are components with their own state and access to services offered by its peers. Since Aspects are implemented as components, they inherit all the benefits of the component model. An Aspect has the ability to create delegates for given interfaces when asked to do so. Interface Factories, which create default instances of particular interfaces, will request delegates from the set of Aspects, chaining the delegates together in series. The resulting wrapped instance will have combined behavior (in the form of a delegate) given by multiple Aspects. Likewise, an Aspect will contribute behavior to multiple interface instances (Figure 4). The choice of Aspects is made dynamically and, when necessary, sent to remote processes.

This is a simple but powerful and dynamic form of AOP. The methods in the interfaces define collections of join-points while any specific Aspect instance implicitly defines a point-cut, depending on which interfaces it chooses to offer delegates for, as well as advice for each relevant join-point (the actual code in the delegate classes). The enclosing Aspect instance provides state as well as dynamic control over the delegation. When a call sequence crosses a host or virtual machine boundary the current list of relevant Aspects can be passed as meta-data to the remote process.

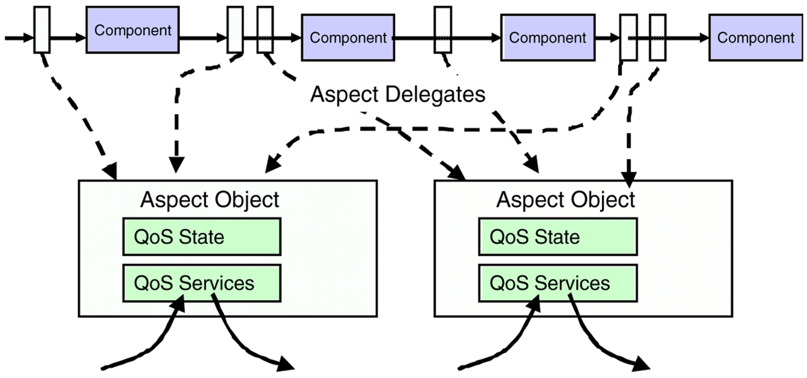


Fig. 4. Aspect Interceptor Pattern crosscuts across multiple service interfaces

Cougaar uses interceptor patterns to open the implementation of several large subsystems, such as the Thread Service and the Metric Service. In addition, the Cougaar's Message Transport System (MTS) has an open implementation, based on the Aspect Interceptor Pattern. The MTS is a set of services designed to allow Agents to communicate via message-passing. The MTS is structured as a predefined series of *stations* through which messages pass on their way from sender to receiver. Each station is defined by an explicit interface and instantiated using a Factory pattern. [15]. MTS Aspect components can attach delegates to one or more station instances at runtime; effectively a simple form of runtime weaving. The MTS Aspect components themselves maintain the state of the collection of delegates they instantiate. This provides the equivalent of a point cut. Finally, by adding meta-data to the messages being passed through the MTS, much like an extensible header [25,26], Aspect behavior can be shared across a distributed system, again at runtime. Section 4 describes several examples of systemic adaptation that use MTS Aspects.

4 Examples of Dynamic Crosscutting

The crosscutting techniques described above were used to add multiple systemic concerns to the Cougaar runtime environment. Most concerns needed to add advice to multiple join points based on the SOA lifecycle and thus one concern could be made up of multiple components, multiple services, and crosscut advice to multiple join points. While many Cougaar subsystems can be extended using the crosscutting techniques, we will concentrate on a single subsystem (MTS), without loss of generality. Over 20 different MTS Aspects have been created to help Cougaar infrastructure handle system concerns for transferring messages over low reliability communication networks. The MTS Aspects can insert behavior at many places along the message processing workflow. Figure 5 shows the base MTS implementation as a series of *stations* where advice can be added. When a station is created all MTS Aspects are informed and interested Aspects can add a delegate to intercept messages as they pass through the workflow. So the chain of delegates is different between each station. This section describes example systemic concerns that were implemented in Cougaar. Some concerns consist of a single component that inserts advice in several places; others consist of multiple components that work together.

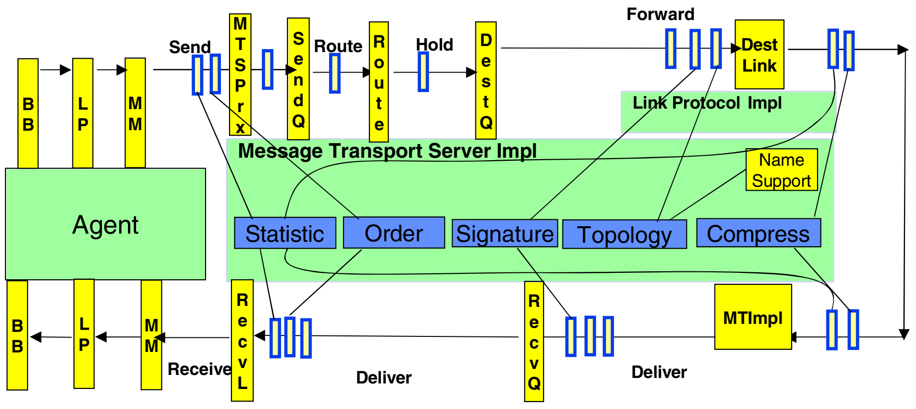


Fig. 5. MTS Aspect can insert advice at many places in the message processing workflow

4.1 Multicast

The multicast example illustrates how a concern needs to insert advice at multiple join points and coordinate between them. The Multicast concern detects the multicast message type and forwards it to all agents in a society. Multicast messages are copied at multiple levels. First the message is sent to all the nodes in the society and then to each agent in the node. Thus, the Multicast concern has to insert itself at multiple MTS stations, to convert message types, to copy messages, look up the addresses of remote nodes at the sender-side and lookup local agents on the receiver-side. Some of these tasks happen when Multicast messages are sent and others when agents register with its node or move to another node. Thus, the Multicast concern crosscuts the station decomposition. On the one hand, multicast is a single, fairly simple, concept. One would expect a good software design for multicast to be implemented in a single class. On the other hand, a typical message-handling system would be decomposed with sending in one class and receiving in another, for all the usual OOP reasons. Since multicast requires changes both on the sender side and receiver side, we can't use traditional OOP to implement it unless we're willing to violate the first point (i.e., keeping the multicast code as a self-contained unit). The Aspect Interceptor technique (Section 3.4) resolves this difficulty. By implementing multicast as an MTS Aspect component, the core message handling code remains simple and stable, while all the multicast code lives in a single place where it's easy to maintain.

4.2 Traffic Masking

The Traffic Masking example illustrates how a concern's advice needs to change based on the runtime situation. The traffic masking concern adds fake traffic between agents to hide the true traffic pattern for security reasons. Traffic Masking must add behavior at multiple MTS stations. Fake messages must be injected at the source agent and thrown away at the destination agent. Also, the traffic between agents must be monitored so that the fake traffic does not overload the connection or interfere with normal traffic. In addition, traffic masking is only enabled under certain situations, based on the type of communication path between agents and security threat level. An example traffic-masking policy may be: under normal security threat level, traffic masking should only be used for communication between nodes that are distributed internationally; but when an insider threat is discovered, traffic masking should also be used on internal LAN traffic. The traffic management functions were implemented using an MTS Aspect that inserted three delegates to generate, monitor, and sink traffic.

One interesting crosscutting interaction happened between the generation and monitoring functions. If the monitor detected that the traffic to an agent was overloaded, the generator would stop generating traffic. The feedback between the monitor and the generator was done through state held within the aspect. When the monitor delegate detected overload, it would set an overload flag. Before sending fake traffic, the generator delegate would check the overload flag and not generate a message if it was set. The traffic generation advice was only executed when enabled by the point cut's dynamic evaluation function.

4.3 Compression

The Compression concern shows how advice needs to be added or removed dynamically based on external factors. Compression of messages is an example of trading off CPU resources for network resources. When the network bandwidth is low, for example from heavy traffic or from using mobile networking links, CPU resources can be used to compress the size of the messages to reduce the amount of data that is being sent. In a sufficiently low-bandwidth environment, a compressed message will be delivered sooner than an uncompressed message, even accounting for the time it takes to compress the message. But in the case of high network bandwidth, compressing the message will make the message be delivered later than an uncompressed message, because compressing takes longer than transmitting the raw message. The status of three external resources must be known in order to determine if compression is useful: the bandwidth of the inter agent communication path, and the CPU capacity of the local and remote hosts.

The join point and the advice are interesting for the Compression concern. Compression itself is done by adding an *OutputStream* to Message serialization. A series of *OutputStream* filters is allowed to transform the message. For example, one Aspect can add an *OutputStream* to count the bytes in the raw message, while another encrypts the message. The order in the serialization pipeline is important. If the byte counting is put last instead of first, it will count the length of the message being sent on the wire instead of the length of the raw message. Also, compression must come before encryption, because an encrypted stream has no patterns that can be compressed. Decompression is likewise added as an *InputStream* in the message de-serialization pipeline.

The problem with adding compression dynamically is that input filters must match the output filters. Given that the sender is dynamically changing the output filters, how does the receiver know which input filters to apply? For example, if the receiver adds a de-compress filter but the sender did not add a compress filter, the resulting messages will be garbled. The solution is to send order of the filters with the message attributes, so that de-serialization can be performed in the proper order at the receiver. The *InputStream* filters are in reverse order of the *OutputStream*, e.g., decrypting before uncompressing. When the message de-serialization join point occurs, the interested MTS Aspects check if their filter is on the list. If it is, they add their InputStream filter to the stream processing chain.

4.4 Gossip

We close our examples with a concern that is implemented as multiple Cougaar components, which must coordinate among themselves and hook into multiple join points. The Gossip concern allows Metrics collected on one Node to be disseminated to other Nodes in the society. The Metrics are transferred by piggybacking Gossip objects as attributes of ordinary messages being sent between Nodes. Gossip takes two forms: requests for Metrics from a neighbor Node, and responses to those requests. Requests are only made for Metrics used within the Node. Also, Gossip is sent only when the value of requested Metrics change. So if no Metrics are requested or the Metrics do not change, no Gossip is sent.

The Gossip implementation takes advantage of special characteristics of both the MTS and the Metrics service. Figure 6 shows the data flow between components that make up the Gossip subsystem. The double-line arrows are the flow of Key Requests and the single-line arrows are Metric Value replies. The Gossip concern consists of four components. The Gossip Aspect handles piggybacking the gossip requests on the inter-node messages. The Gossip DataFeed is a plugin to the Metric service and publishes metrics into the local metrics service. The other two Aspects set thresholds on which metrics to gossip and how often.

Notice that some Gossip components add new services to the system (Value Qualifier Aspect, Key Qualifier Aspect, and Gossip Data Feed). Other Gossip components can lookup these services at load time to integrate the components. The Gossip Aspect adds delegates to the MTS stations to gain access to messages flowing between agents. Gossip components are added at multiple insertion points in the Cougaar configuration. A configuration rule for the Gossip concern specifies which Gossip component to load and in what order.

The Gossip concern shows the power of the Cougaar crosscutting techniques. The whole subsystem can be added or removed based on a single rule at load-time. Advice is added at different subsystems, such as the Metric service and the Message transport service. The concern has access to join points as low-level as intercepting messages and as simple as adding or requesting a service.

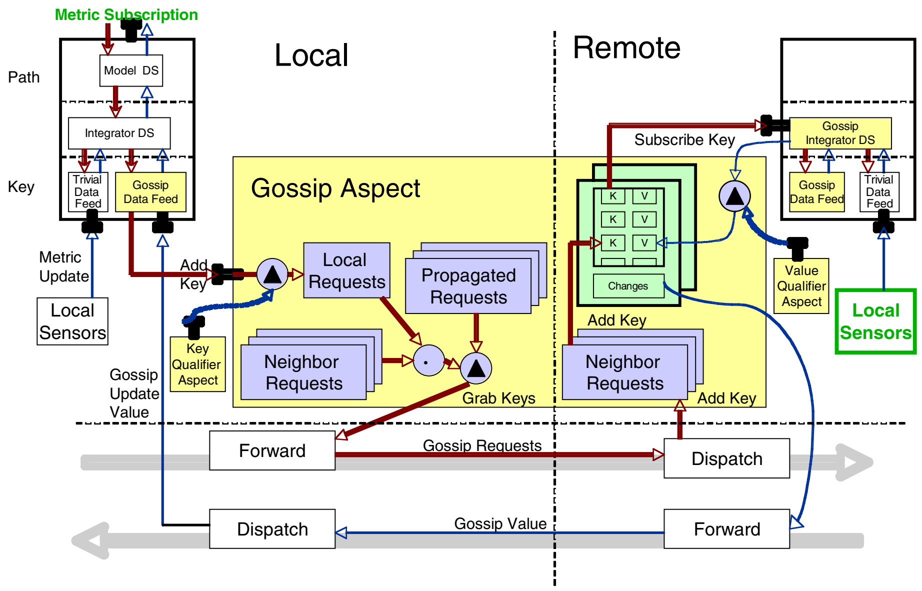


Fig. 6. Gossip system disseminates Metrics data between Cougaar Nodes, by piggybacking the data and request on normal message traffic

5 Relative Overhead of Crosscutting

The following experiment illustrates the relative cost of using Cougar crosscutting techniques to add adaptive behavior to an application. The experiment consists of running a simple application under different configurations, varying the amount of adaptive code loaded and the configuration of hardware resources. Note, we do not show explicit measurements of the overhead associated with the crosscutting mechanisms described in Section 3. This overhead is trivial and only adds a delegated call to each service interaction. We do show the overhead relative to the adaptive behavior and the application behavior. That is, what is the cost of checking if adaptive advice should be added and the cost of executing that advice, verses the cost of executing the application behavior. The conclusion is that the relative cost of the overhead is more important than the absolute cost. The situations in which crosscutting overhead is problematic are also the situations in which adaptive behavior is not needed, and crosscutting is therefore not used.

To understand the performance of the application we must look at a total system view. The application business behavior is the functional purpose of the system and it must run in all configurations. The adaptive advice depends on the situation: it will run in some configurations but not others. Since Cougaar is adapting at runtime, we must also consider the cost of point cut evaluation. Usually, the time to evaluate the point cuts is much smaller than the time to run the adaptive advice: when the advice runs, the point cut evaluation is relatively small. The problematic case is when the point cut evaluation selects no advice [33]. Cougaar gets around this problem *implicitly* in most cases by performing runtime point cut evaluations only for join points that potentially will have advice. The reason is that Cougaar AOP techniques limit potential join points to life-cycle events of service interfaces. In addition, the point cut evaluation is phased across several life-cycle events. For example, concerns add delegates at service resolution time only if they plan to add advice.

The benefit and overhead of an individual service depends on the overall configuration of a Cougaar society. For example, a service in one configuration could offer little benefit and have a high overhead relative to other components. This service is a candidate for performance improvement, or even removal from the configuration. Conversely, in another configuration, the same service may have a high benefit, with only a small overhead relative to other components, making the service critical to the society. Node-level QoS-adaptation services tend to have the following overhead behavior: high benefit in a certain situation, but not in others. Local vs. distributed configurations are good examples of this. A major feature of Cougaar is that QoS-adaptation can be removed for local configurations and added back for distributed configurations. (Processing in a local situation does not usually need QoS-adaptation because the network and host resources do not change.)

We will use a Ping society to illustrate the effect of adaptive code on the performance of an application. The Ping society consists of two agents that transfer objects between their blackboards. The Source agent writes an object onto its blackboard after which the Cougaar infrastructure makes a copy of the object on the Sink's blackboard. The Sink agent is triggered by a callback when the object appears on its blackboard, after which it writes a new object back to the Source. In a real world application, business logic would run after the ping is received at the Source and Sink.

But for the sake of the illustration, the ping application has zero business logic processing. If there was more business processing, the relative overhead of the crosscutting mechanisms would be even less.

For the single ping loop, all processing associated with the ping is executed serially. When a systemic component is added, any point cut overhead and advice processing will add latency to the ping loop. The performance impact can readily be measured, simply by measuring the rate of pings per second.

Table 1. Relative Overhead of Node-Level Services (Single Ping, 3Ghz Processor)

Configuration			Performance (Pings/second)		
Serial	**RMI**	**2 Hosts**	**Standard**	**Minimal**	**%**
			1061	3229	304%
X			306	466	152%
X	X		139	165	121%
X	X	X	179	210	117%

In local configurations, the overhead for a QoS-adaptive code is relatively high. Table 1 illustrates the differences in societal configuration overhead. The first row shows the relative cost of standard node-level services. The standard Ping society uses adaptive Metric and Thread services. When these services are removed in the minimal Ping Society, the performance increases by a factor of three. If the Metrics Service and Thread Management are unnecessary, these services can and should be removed at configuration time.

In distributed configurations, message serialization dominates overall society performance. QoS-adaptation services become necessary to react to changes in the networked resources, such as device failure, competing network traffic, and security attacks. The relative overhead of these QoS services is small. Table 1 presents the overhead associated with message serialization in the context of performance. Row 2 of Table 1 shows a performance decline associated with serialization alone. These results reflect an additional test component in our minimal Ping society, one which forced the serialization of messages even when sent locally. This resulted in a performance decrease of more than a third. Row 3 shows the combined overhead of using RMI and serialization, when the source and sink agents were separated onto two nodes running on the same host. Notice that there is almost a factor of ten drop in performance when sending pings between node processes. Despite this, the relative performance increase of removing the metric and thread services is a trivial 21%. Since the benefit of these services is high and the relative overhead is low, this simple test suggests that these QoS-adaptive services should be installed in distributed configurations.

The last row of Table 1 illustrates how the performance actually increases slightly when the nodes are run on two hosts connected by a high-speed network. The increase is due to the fact that some of the society processing (e.g. Metric Service statistics processing) can be done in parallel with the ping loop.

6 Related Works

Cougaar is one of many distributed system frameworks that support runtime adaptation. Most major frameworks now recognize the usefulness of component-based architecture and some are even supporting AOP techniques. We describe in this section a variety of such frameworks, which meet similar crosscutting requirements and compare their approaches with those of Cougaar. The frameworks can be divided into three broad classes. Extensions to CORBA-based middleware, such as CORBA Component Model (CMM) [8] and OIF [11], allow end-users adaptation only between clients and servers that use CORBA to intercommunicate. Component-based middleware, such as IONA ACT [24], use components and services to build up higher level communication schemes within the middleware framework itself. Object-based AOP frameworks, such as JBoss AOP [17, 22] and JAC [32], Aspectwerkz [34] and Spring [35] allow end-users to add fine grain advice anywhere inside the middleware implementation.

6.1 CORBA Based Middleware

CORBA 3.0 extends the CORBA 2.0 distributed object model to define a CORBA Component Model (CCM). The CCM specification provides a standard way of designing components, configuring the connections of these components and their default attributes at assembly time, packaging these components as distributable units, and deploying them over the network. The QuO framework was extended to work with CCM by encapsulating QoS adaptive behaviors as CCM components [18]. These adaptive components, called a *qosket* components, can be developed separately from functional components, can be configured with the application components using CCM tools, and can adapt the behavior of the system at runtime. A *qosket* component implements QoS-adaptation and is inserted into data-flow between two components using a specification, creating a chain of responsibility pattern. The specification of this chain is statically defined in the deployment configuration. Point cut evaluation happens at both deployment-time and interface invocation-time. At deployment time the deployment spec determines which qosket components to load and the contracts contained in those qoskets determine which advice to assert based on system conditions. A qosket component is limited to wrapping a single interface between CMM components, and does not support Cougaar's Aspect interceptor technique.

Object Infrastructure Framework (OIF) [11] has a similar idea of injecting behavior between clients and servers in a CORBA-based system. OIF is based on CORBA 2.0, so does not benefit from the CMM. OIF can generate proxies for both the client and server sides. OIF can compose multiple "-ilities" concerns into a chain of responsibility pattern which is code generated into a single pair of proxies. The internals of the proxies can have adaptive behavior, but OIF lacks the structured assessment of the current situation as supported in QuO contracts and System Condition evaluators.

6.2 IONA Adaptive Runtime Technology

IONA's Adaptive Runtime Technology (ART) is a low-level infrastructure on which several middleware frameworks are built, such as CORBA POA, J2EE EJB and servlet containers [24]. IONA ART consists of a set of communication and storage services implemented as components. The Chain of Responsibility Pattern [13] is used extensively to interconnect the components. Interceptors are also exposed so that developers can extend the framework. ART does not have a mechanism similar to Cougaar Binders, but wrapper delegates can still be added using the interceptor mechanisms.

6.3 Byte-Code Manipulation

Several JAVA frameworks [17, 22, 32, 34, 35] have integrated in byte-code manipulators that allow AOP to be applied at the object-level, without extending the Java language [30]. Aspects can be deployed and un-deployed at runtime, and use Java classes as aspects. For example, JBoss is a J2EE web-services framework and comes with a prepackaged set of aspects that are applied via annotations, point cut expressions, or dynamically at runtime [22]. These frameworks work by overriding class implementations at load-time, specifying 'Interceptor' classes that wrap around classes and use reflection access to method behavior [30]. In addition JBoss can form a chaining mechanism between these Aspects, analogous to the Cougaar Aspect Interceptor Pattern (Section 3.4). These AOP frameworks can override any class in the system, hence offers a finer granularity of join points than Cougaar, which has course grain join points based on the service life-cycle of the SOA. AOP frameworks have the advantage of being able to access the internal structure of components and add advice, whereas Cougaar can only offer advice at explicitly exposed interfaces to its services. The advantage of Cougaar's service-boundary crosscutting is that it is defined at the natural boundary between Cougaar components, the service interface, and can be selectively enabled and disabled on a per-component basis instead of a per-class basis.

7 Conclusions

Cougaar supports a spectrum of crosscutting techniques that can be used to implement runtime adaptation. These techniques were successfully used to create a survivable infrastructure composed of independently developed systemic concerns, such as security, robustness, bandwidth management, and storage management. These survivability concerns can be both statically configured at society load time and dynamically configured, enabled, and disabled at runtime. Although Cougaar's crosscutting techniques are restricted to join points defined by component life-cycle and service method boundaries, our experience has shown that these join points were sufficient to meet our target application's complex configuration and adaptation requirements.

References

1. Bergmans, L., Aksit, M.: Composing Multiple Concerns Using Composition Filters, Communications of the ACM, special issue on AOP (October 2001)
2. Lieberherr, K., Ovlinger, J., Mezini, M., Lorenz, D.: Modular Programming with Aspectual Collaborations. College of Computer Science, Northeastern University, Tech report NU-CCS-2001-04 (March 2001)

3. Ossher, H., Tarr, P.: Using Multidimensional Separation of Concerns to Reshape Evolving Software. Communications of the ACM, 43–50 (October 2001)
4. Ossher, H., Kaplan, M., Katz, A., Harrison, W., Kruskal, V.: Specifying Subject-Oriented Composition. Theory and Practice of Object Systems 2(3) (1996)
5. Kiczales, G., Lamping, J., et al.: Aspect-oriented programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, Springer, Heidelberg (1997)
6. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Grisold, W.: Getting Started with AspectJ. In: CACM, p. 59 (October 2001)
7. Cougaar Distributed Agent System, open source at http://cougaar.org
8. Zinky, J., Bakken, D., Schantz, R.: Architectural Support for Quality of Service for CORBA Objects. Theory and Practice of Object Systems (April 1997), http://quo.bbn.com
9. Kiczales, G.: Beyond the Black Box: Open implementation. IEEE Software (January 1996)
10. Shapiro, R., Zinky, J., Rupel, P.: The Aspect Pattern. In: OOPSLA 2002 Workshop on Patterns in Distributed Real-time and Embedded Systems, Seattle, Washington (November 2002)
11. Filman, R., Stuart, B., Lee, D., Linden, T.: Inserting Ilities By Controlling Communication. Communications of the ACM (January 2002)
12. Cable, L.: Extensible Runtime Containment and Server Protocol for JavaBeans Version 1.0. JavaBeans Glasgow Specification (December 1998)
13. Vinoski, S.: Chain of Responsibility, IEEE Internet Computing (November/December 2002)
14. Schmidt, D., et al.: Pattern-Oriented Software Architecture: Patterns for Concurrency and Distributed Objects, vol. 2. John Wiley and Sons, New York (2000)
15. Gamma, E., et al.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading Mass (1995)
16. Schweisguth, D.: Second-generation aspect-oriented programming, Java World (July 2004)
17. Yuan M: On the the Road to simplicity: JBoss 4.0 simplifies middleware development, Java World (February 2005)
18. Sharma, P., Loyall, J., Heineman, G., Schantz, R., Shapiro, R., Duzan, G.: Component-Based Dynamic QoS Adaptation in Distributed Real-time and Embedded Systems
19. Helsinger, A., Kleinmann, K., Brinn, M.: A Framework to Control Emergent Survivability of Multi Agent Systems. In: Kudenko, D., Kazakov, D., Alonso, E. (eds.) Adaptive Agents and Multi-Agent Systems II. LNCS (LNAI), vol. 3394, Springer, Heidelberg (2005)
20. Ultralog Project, http://dtsn.darpa.mil/ixodarpatech/ixo_PrintFeatureDetail.asp?id=61
21. Quality Objects Project (QuO), http://quo.bbn.com
22. JBoss AOP Users Guide, http://labs.jboss.com/jbossaop/docs
23. JBoss AOP Reference Manual, http://labs.jboss.com/jbossaop/docs
24. Orbix 6.0 Introduction Manual, http://www.iona.com/
25. Braden, R., Faber, T., Handley, M.: From Protocol Stack to Protocol Heap – Role-based Architecture, HotNets I, Princton NJ, USA (October 2002)
26. Filman, R.: Injectors and Annotations. In: Magnusson, B. (ed.) ECOOP 2002. LNCS, vol. 2374, Springer, Heidelberg (2002)
27. Bers, J., Redi, J.: Supporting Robot Teams with CougaarME over Wireless Ad-hoc Networks, 1st Open Cougaar Conference, New York, NY (July 2004)
28. Roll, W: Towards Model-Based and CCM-Based Applications for Real-Time Systems. In: ISORC 2003. Proceedings Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, IEEE Computer Society Press, Los Alamitos (2003)

29. Filman, R., Friedman, D.: Aspect-Oriented Programming is Quantification and Obliviousness. In: OOPSLA 2000. Proceedings of workshop on Advanced Separation of Concerns (2000)
30. Zdun, U.: Pattern language for the design of aspect languages and aspect composition frameworks. IEE Proceedings Software (April 2004)
31. Zinky, J., Loyall, J., Shapiro, R.: Runtime Performance Modeling and Measurement of Adaptive Distributed Object Applications. In: DOA. Proceeding of International Symposium on Distributed Object and Applications (October 2002)
32. Pawlak, R., Seinturier, L., Duchien, L., Florin, G.: JAC: a flexible framework for AOP in Java. In: Yonezawa, A., Matsuoka, S. (eds.) Metalevel Architectures and Separation of Crosscutting Concerns. LNCS, vol. 2192, Springer, Heidelberg (2001)
33. Hanenberg, S., Hirschfeld, R., Unland, R.: Morphing aspects: incompletely woven aspects and continuous weaving. In: Proceedings of the 3rd international conference on Aspect-oriented software development (March 2004)
34. Aspectwerkz, http://aspectwerkz.codehaus.org/
35. Spring AOP Framework, http://www.springframework.org/
36. Filman, R., Lee, D.: Redirecting by Injector. In: International Conference Distributed Computing Systems Workshop (April 2001)
37. Zinky, J., Shapiro, R.: The Aspect-Oriented Interceptors Pattern for Crosscutting and Separation of Concerns using Conventional Object Oriented Programming Languages. In: The 2nd AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS), part of International Conference on Aspect-Oriented Software Development, Boston (March 2003)

Property-Preserving Evolution of Components Using VPA-Based Aspects[⋆]

Dong Ha Nguyen and Mario Südholt

OBASCO project; EMN-INRIA, LINA
Dépt. Informatique, École des Mines de Nantes
4 rue Alfred Kastler, 44307 Nantes cédex 3, France
{Ha.Nguyen, Mario.Sudholt}@emn.fr

Abstract. Protocols that govern the interactions between software components are a popular means to support the construction of correct component-based systems. Previous studies have, however, almost exclusively focused on static component systems that are not subject to evolution. Evolution of component-based systems with explicit interaction protocols can be defined quite naturally using aspects (in the sense of AOP) that modify component protocols. A major question then is whether aspect-based evolutions preserve fundamental correctness properties, such as compatibility and substitutability relations between software components.

In this paper we discuss how such correctness properties can be proven in the presence of aspect languages that allow matching of traces satisfying interaction protocols and enable limited modifications to protocols. We show how common evolutions of distributed components can be modeled using VPA-based aspects [14] and be proven correct directly in terms of properties of operators of the aspect language. We first present several extensions to an existing language for VPA-based aspects that facilitate the evolution of component systems. We then discuss different proof techniques for the preservation of composition properties of component-based systems that are subject to evolution using protocol-modifying aspects.

1 Introduction

Interaction protocols are a popular means to construct correct component-based systems and document them (see, *e.g.,* [8,17,23]). A major question for the evolution of component-based systems is whether evolution preserves compositional properties of these systems, in particular compatibility and substitutability of components, two fundamental notions that are typically defined in terms of subset relationships of trace sets (and sometimes failure sets) admitted by the original and evolved versions of a system [15,23]. Currently, almost all component-based systems with interaction protocols have used finite-state protocols, including all of the previously cited ones. Recently, however, approaches using more expressive non-regular protocols have been proposed [3,19].

[⋆] Work partially supported by AOSD-Europe, the European Network of Excellence in AOSD (www.aosd-europe.net).

If software components are equipped with interaction protocols, evolution of component-based systems can frequently be expressed in a concise manner using aspects that modify executions that have to conform to such protocols. In the last couple of years different researchers have considered protocol-modifying aspect languages, notably [4,6,14,22]. However, none of these approaches has explored the preservation of compositional properties of software components in the context of aspects modifying such interaction protocols. Among the very few that have touched on this question, Farías [7] proposes the extension of the language of regular aspects by protocol-modifying operators and considers proof techniques for the resulting finite-state based aspects.

We consider how compositional properties can be defined and verified in the context of the evolution of components that are equipped with a more expressive brand of interaction protocols, protocols defined in terms of Visibly Pushdown Automata (VPA) [2]. VPA allow to define protocols that include well-formed nested contexts, such as correct nesting of recursive calls to and returns from a server. VPAs are strictly more expressive than finite-state automata (which generate regular languages) but strictly less so than pushdown automata (which generate context-free languages). In contrast to finite-state based systems, VPA-based protocols allow (some) nested terms of, *e.g.,* call and returns, to be correctly matched without having to restrict the nesting depth. In contrast to pushdown automata, VP languages are closed under all basic operations, including intersection and complement, and all basic decision problems are decidable.

In this paper, we present two main contributions. First, we present four extensions to the language of VPA-based aspects [14] that are useful in the context of component evolution for distributed applications: a more general definition of sequence pointcuts, a new pointcut operator that allows nested contexts to be matched if their depths exceed a threshold, a permutation for well-balanced contexts and a new advice construct that allows to close an open nested context, *e.g.,* for error handling purposes. We motivate how these extensions can be used in the context of the evolution of software components for the implementation of distributed search algorithms typical, *e.g.,* for P2P applications. Second, we show how compatibility and substitutability properties of component-based applications can be proven if interaction protocols are subject to evolution using VPA-based aspects. We present, in particular, how some composition properties can be handled fully in terms of the properties of protocol-operators even in the presence of aspects.

The paper is structured as follows. Sec. 2 motivates using VPA-based aspects to define evolution of distributed components with explicit protocols and to verify the preservation of compositional properties for those systems. Sec. 3 presents the VPA-based aspect language and defines the four extensions to the language. Sec. 4 presents the proof techniques for the preservation of composition properties in the presence of aspects. In Sec. 5 we discuss related work before concluding in Sec. 6.

2 Motivation

The large majority of component-based systems is based on components whose interfaces define sets of (method or function) signatures that correspond to services provided by the component. Furthermore, signatures that correspond to services that a component requires to be provided from other components are also frequently declared as part of component interfaces. Such interfaces do not, however, specify any information about the semantics of provided and required services. With such a notion of components, component evolution therefore has to be defined essentially on the level of component implementations and component properties, such as compatibility (that ensures that two components communicate correctly) and substitutability (that ensures that a component can be substituted without problems for another one in arbitrary usage contexts), can only be defined in terms of service signatures, *i.e.,* not including any guarantees on the effective behavior of the components.

Protocols that govern component interactions and that are explicit in component interfaces have been proposed as a means to overcome these limitations. Changes to interaction protocols allow modelling component evolution in terms of modifications because they enable the specification of new restrictions on previously enabled communications between components. Furthermore, compatibility and substitutability properties can be formally based on the interaction protocols. Interaction protocols have mostly been defined in terms of finite-state automata (*e.g.,* [8,17,23]), only a few have considered more expressive non-regular protocol languages (*e.g.,* [3,19]). None of these approaches, have considered component properties in the context of dependencies involving recursive computations, which are, however, crucial to a large class of application domains.

In order to motivate the special requirements of such application domains, let us consider a quite typical representative, P2P algorithms, in particular, the approach to unstructured P2P networks proposed by Lv et al. [13]. A central proposal of this work is a particular algorithm to search a decentralized and unstructured P2P network. The basic algorithm can be summarized as follows. When a peer initiates a search to find a document, it sends a corresponding query message to (some of) its neighbors in some order. When a peer receives a query message it first checks its local store and replies to the query with the corresponding file if it is found locally. Otherwise, it forwards the request to its neighbors. To ensure termination, this search protocol makes use of a time-to-live (TTL) value in order to limit the search space. This value is decremented at each peer that is visited and the search is stopped when 0 is reached.

From a component point of view the behavior on a node we consider can be characterized by the incoming and outgoing communication events as shown in Fig. 1 that can be seen as defining its signature-based interface. The P2P algorithm most basically consists of nodes sending out `query` messages and receiving `reply` messages from its neighbors. Furthermore a node may `abort` on-going queries if requested to do so (`abortRequest`). Finally, a node may `join` or `quit` the P2P network.

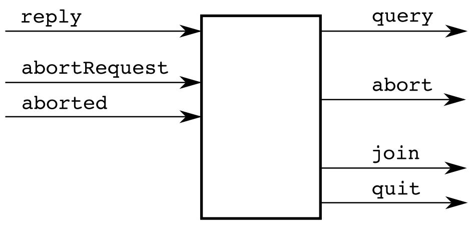


Fig. 1. Component interface (service part) for P2P algorithms

All of the (regular or non-regular machine based) approaches to interaction protocols cited above do not permit to express (many) fundamental protocols that involve the recursive nature of the search algorithm in the above P2P context. Furthermore, none of those allow to reason about the properties underlying such protocols. VPA-based protocols allow to directly define many of such protocols and enable formal reasoning about correctness properties for components defined in terms of them. Finally, aspects over interaction protocols allow component evolution to be expressed naturally; VPA-based aspects enable evolutions including modifications to such recursive behaviors to be defined and the verification of the preservation of composition properties in such cases.

To illustrate the basic advantage of VPA-based aspects over regular aspects, consider the (regular) protocol for aborting queries that is shown in Fig. 2. If an abort request occurs a query should be aborted and new queries only be enabled after abortion has been performed. With aspects, abort requests can be performed by triggering an advice *abort* when the execution event *abortRequest* occurs: we note such a basic aspect as $abortRequest \rhd abort$. The (complete) regular aspect shown in the figure allows these interactions; however, it does not enforce the restriction that abort requests should only be allowed if there is at least one on-going query. This is a reasonable constraint if abortion is an operation which might consume much time or other resources. Moreover, the protocol in Fig. 2 also allows the number of replies occurring at state 1 to exceed the number of queries (which is obviously an unreasonable situation).

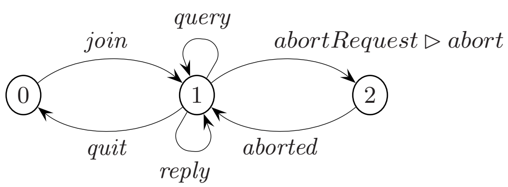


Fig. 2. Aborting on-going queries (regular)

In contrast to finite-state automata, VPA have a stack which can be used to distinguish different calls to the same method. This way, the constraint of allowing abort requests only if there are on-going queries can be expressed, *e.g.,* using the VPA shown in Fig. 3. The main characteristics of VPA is that calls and returns are distinguished by well-distinguished push and pop symbols on a stack, and a return transition can only be performed if the symbol of a matching call is on top of the stack. Call and return operations in the protocol are indexed with symbols which are pushed or popped on the VPA stack; furthermore return operations are underlined. In the VPA example in Fig. 3, the stack symbols allow to distinguish the first query operation from the remaining ones and the VPA behavior thus ensures that in state 2 only a number of replies can be performed that equals the number of queries done *at state 2* (*i.e.,* with index q), while the query with index fst performed between states 1 and 2 remains on the stack: therefore in state 2 there is always at least one query on the stack and abort requests are always made in an appropriate state. Basically, the example illustrates that VPA allow to "count" calls and allow to require that the number of returns match the number of calls.

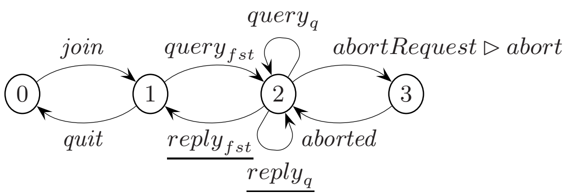


Fig. 3. Aborting on-going queries (VPA)

Finally, note that typical modifications to such recursive algorithms, such as stopping a recursive algorithm at a given call depth, cannot be directly expressed using regular structures: they can only be expressed using finite-state structures by fixing the maximum number of open recursive calls. Such a constraint is, however, (i) unpractical in application contexts such as P2P networks where the recursive depth to which a network is to be explored may be rather large and (ii) signifies that no general composition properties, *i.e.,* that do not include such limits on the recursion depth, can be proven for regular aspects. VPA-based aspects provide a solution for these two problems for a large set of recursive interaction protocols.[1]

3 Extending VPA-Based Aspects for Component Evolution

In this section we present the VPA-based aspect language that we propose for component evolution. After presenting the main characteristics of the language by a small example, we first introduce the pointcut and advice operators that we have introduced to cope with component evolution; this part focuses on the four operators that are new compared to our previous proposal [14]. We then present the formal definition of the complete aspect language and give an informal account of the semantics of the language.

[1] Since VPA are less expressive than pushdown automata not all context-free protocols can be represented, though.

3.1 VPA-Based Aspects by Example

In the motivation, we have presented an aspect to `abort` search queries after an `abortRequest` message has been received. The VPA-based aspect that ensures that aborts are only performed if at least one search query is active, can be defined using our language as follows:[2]

$$
\begin{aligned}
join \;;\; \mu b.\; query_{fst} \;;\; \mu c.(&\underline{reply_{fst}} \;;\; b)\\
\Box\; &(query_q \Box \underline{reply_q} \;;\; c)\\
\Box\; &(abortRequest \triangleright abort \;;\; c)
\end{aligned}
$$

As common for aspect languages, the above aspect definition essentially consists of a pointcut definition, which defines the sequences of execution events the aspect matches, and an advice definition, which defines the actions to be executed once a match occurs.

In the above aspect definition, the *pointcut* specifies sequences of events of interest for components providing services for P2P search protocol as illustrated in Fig. 1. Repetitions in a pointcut are defined using the 'μ' operator, bind a recursion variable (b in the outermost repetition above) and extend to the next occurrence of the recursion variable. Sequences of events are formed using the sequencing operator ';'. A choice among two or more matching events is expressed by operator '$\Box$'. For instance, "$\mu c.\; query_q \;\Box\; reply_q \;;\; c$" presents a pointcut that repeatedly matches *query* or *reply* events. Furthermore, VPA call as well as return events are indexed with stack symbols; VPA returns are underlined. For example, $query_{fst}$ and $reply_{fst}$ will match only the first query and its corresponding last reply of the search process.

The above aspect contains only one *advice*, the call to *abort* in the final branch of the choice that contains the basic aspect $abortRequest \triangleright abort$ consisting of a single event triggering the advice.

3.2 Pointcut and Advice Operators for Component Evolution

Calls and corresponding returns that are used in pointcuts of VPA-based aspects enable well-balanced contexts to be used in interaction protocols, matched during program execution and statically analyzed, *e.g.,* for conflicts with other VPA-based aspects. Such aspects may be concisely defined in terms of operators that are specialized to well-balanced contexts. In the following, we introduce four operators (three pointcut operators and one advice operator) that extend our previous language and show how these operators benefit component evolution in the context of P2P systems.

[2] Note that this language is not intended for programming at the user-level but rather a means to support the formal definition of its semantics and property analysis for aspects and components. The integration into mainstream programming languages as well as corresponding implementation support is discussed in [14].

Depth-dependent operators. In addition to restricting the depth of recursion as above, evolution of P2P distributed algorithms often aims at the optimization of the underlying traversal strategy through heuristics to perform a more superficial but faster search on nodes whose distance from the root node exceeds a certain threshold. Since VPA faithfully allow to define the depth of nested terms, such heuristics can be directly expressed using a pointcut operator $D_m^{>k}$ that matches only calls to m that occur at a depth larger than k. For example, the following aspect caches queries at depth greater than 5 (where $\mu a.\ldots; a$ denotes recursion in VPA-based aspects):

$$\mu a.\ D^{>5}_{query_q} \rhd getCacheValue\ ;\ a$$

General sequencing operator. Aspect languages for protocols typically include a sequence operator ; on the pointcut level. Sequences of events in pointcuts may, however, match executions according to different semantics. Most frequently, the sequence $a; b$ matches an execution trace that contains the event a followed by a sequence of arbitrary events except b followed by the event b. Using this semantics — which has been pioneered by the approach of stateful aspects [4] and used in numerous others (*e.g.,* JAsCo [21] as well as our previous approach to VPA-based aspects [14] — the aspect

$$join\ ;\ \mu a.\ query_q \rhd saveContext\ ;\ a$$

waits for the current node to join a network and then repeatedly saves the local context (*e.g.,* on a backup server) if a query occurs.

This sequencing operator does not fit some common situations in evolution scenarios. Consider the two following cases:

– Components are extended by a backup operation that makes superfluous the (potentially costly) context saving but only if the backup is executed immediately after the query operation.
– General query operations must not be performed in certain cases, *e.g.,* if an erroneous situation occured or a certain neighbor has to be excluded from the search.

These scenarios are instances of two general problems: the sequence operator introduced above does not allow us to define that occurrences of events have to immediately follow a particular event (even if the pointcut language includes a negation operator). Furthermore, it is tedious to exclude traces by excluding specific sets of individual events that are not allowed to occur.

In order to allow the concise definition of evolutions of the such kind, *i.e.,* by arbitrary interleaving as well as through the (mandatory) absence of interleaving we propose a general sequencing operator $;_{\mathcal{I}}$ where $\mathcal{I}$ specifies the set of events that may be interleaved between the argument events ($\mathcal{I}$ may be $\emptyset$ to exclude any interleaving or the set of all events to allow arbitrary interleaving).

Permutation operator for well-balanced contexts. Permutation operators are frequently used for the construction of protocols that allow the arbitrary interleaving of sets of events. In the presence of well-balanced contexts, interleavings of calls as well as calls and corresponding returns are subject to restrictions that cannot be modeled simply using the standard permutation function that generates all permutations.

In P2P networks, for instance, a query on one node that triggers a query on a neighbor, *e.g.,* q_1 followed by q_2, must be followed by replies in the reverse order $\underline{r_2}$, $\underline{r_1}$. The permutation $q_1, q_2, \underline{r_1}, \underline{r_2}$ is not valid.

An advice operator to close open calling contexts. One major characteristics of a P2P network is the dynamism of peers. They can come and go unpredictably and thus it is common to have queries without corresponding replies. Subnetworks may be disconnected or messages lost. Error handling for those situations may involve the introduction of events that close a number of open recursive calls in order to skip the traversal of part of the underlying distributed network in which an error occured. Using VPA-based aspects such error handling strategies can be expressed using the advice-level operation $closeOpenCall_m$ that closes the open call to m: pointcuts matching on nested contexts can then be used to restrict the application of such advice to appropriate parts of the network. The following example illustrates the use of a closing operator to add a number of "fake" replies to queries when the query exceeds a given connection timeout (where $\Box$ denotes the choice between alternatives):

$$\mu a.\ query_f; (\underline{reply_f} \ \Box\ (connectionTimeOut \triangleright closeOpenCall_{query_f}))\ ;\ a$$

3.3 Syntax

Figure 4 presents the complete syntax of VPA-based aspects. (The operators introduced above that extend the language of [14], are marked by boxes.)

A represents aspects which are defined by VPA-based expressions over basic aspects $P \triangleright Ad$ where P is a pointcut, and Ad is an advice action.

A pointcut P is constructed from terms T denoting method calls or returns or local operations (that may not influence the stack) as well as conjunctions and complements of terms. As discussed above, the sequencing operator $;_{\mathcal{I}}$ is the general sequencing operator where $\mathcal{I}$ specifies the type of events allowed to interleave between two other events. Furthermore, pointcuts allow regular expressions of events to be matched (remember that VPA are strictly more expressive than finite-state automata). A pointcut can also be defined using depth-dependent operators (non-terminal D). Three different such operators are provided: D_M^{int} (nested execution to a specific depth), $D_M^{\leq int}$ (nested execution for with depths inferior to an integer value) and $D_M^{>int}$ (nested execution for depths greater than an integer value). Finally, pointcuts may be constructed using the two permutation operators $perms_{nested}(lst) and perms_{flat}(lst)$ that, respectively, allow to express concisely permutations concerning nested and flat pairs of calls and corresponding returns. These two constructors have been shown because its frequent use as a building block for VPA-based protocols. Other permutation operators specific for well-balanced contexts can be defined but are less useful.

$$
\begin{array}{lll}
A & ::= & \mu a.A \\
 & | & \boxed{P \, \triangleright \, Ad \, ;_{\mathcal{I}} \, A} \\
 & | & \boxed{P \, \triangleright \, Ad \, ;_{\mathcal{I}} \, a} \\
 & | & A \, \Box \, A \\
 & & \\
P & ::= & T \ | \ D \ | \ \boxed{Perms} \\
 & | & \boxed{\mathrm{P} \ ;_{\mathcal{I}} \ \mathrm{P}} \ | \ P \| \, P \ | \ P\{\mathrm{int}\} \ | \ P+ \ | \ P* \\
T & ::= & Tl \ | \ !T \ | \ T \ and \ T \\
Tl & ::= & M \ | \ M_{Id} \ | \ \underline{M_{Id}} \\
D & ::= & D_M^{int} \ | \ D_M^{\leq int} \ | \ \boxed{D_M^{>int}} \\
Perms & ::= & \boxed{perms_{nested}(lst) \ | \ perms_{flat}(lst)} \\
 & & \\
Ad & ::= & send(M, Id) \ | \ \boxed{closeOpenCall(M, int)} \ | \ Ad \ ; \ Ad \\
 & & \\
M & ::= & \mathrm{const} \ | \ \mathrm{var} \qquad \text{// method names} \\
Id & ::= & \mathrm{const} \ | \ \mathrm{var} \qquad \text{// stack, component ids} \\
lst & ::= & \mathrm{list[M]} \qquad \text{//list of methods}
\end{array}
$$

Fig. 4. Syntax of aspects over VPA-based protocols

Advice *Ad* consists of finite sequences constructed from the operator for closing call contexts *closeOpenCall* and calls to services of named components. An application of the operator $closeOpenCall(m, n)$ inserts n calls to the return instruction corresponding to the call m.

3.4 Semantics

In this section, we informally present how the language extensions introduced in this paper can be integrated in the formal framework for the definition of small-step operational semantics for VPA-based aspects introduced in [14]. To this end, we first give a brief overview of that framework and then explain how to define the four new operators using this framework. Here in this paper, we will mainly discuss the semantics of the new extensions introduced in the previous subsection.

Overview of formal framework. The formal framework introduced in [14] defines the semantics of VPA-based aspects as a small-step semantics of the execution of woven program. Aspects (non-terminal A in Figure 4) are interpreted by repeatedly matching events of the execution of the base program with basic aspects $p \triangleright a$. If the pointcut p matches, the advice a is executed and the next basic aspect defined using the repetition, sequence and choice operators is determined.

Pointcut declarations are translated into a pointcut VPA (in the sense as defined by Alur and Madhusudan [2]). This pointcut VPA is then used during application of basic aspects to decide matching of, in particular, stack-manipulating calls and returns. Complex pointcuts that do not match individual execution events, such as regular expression pointcuts, correspond to paths in the pointcut VPA. As part of the operational semantics, program configuration containing the state of the pointcut VPA that evolves along with the base program execution.

Advice is inserted once the corresponding pointcut has matched by inserting the advice body — sequences of stack-manipulating return operations defined using the advice operator *closeOpenCall* or calls to component services — before, after or instead of the matched base execution event.

Definition of language extensions. In the remainder of this section we present how the extensions of the VPA-based language introduced in this paper can be defined using this formal framework.

Permutation operators. The two permutation constructors generate sequences of pairs of well-balanced pairs of calls and corresponding returns.

- The constructor $perms_{nested}(lst)$ generates nested pairs of call and return events. $perms_{nested}(query_a, query_b)$, *e.g.,* generates

  $$(query_a \ ; \ query_b \ ; \ \underline{reply_b} \ ; \ \underline{reply_a}) \ \Box \ (query_b \ ; \ query_a \ ; \ \underline{reply_a} \ ; \ \underline{reply_b})$$

- The constructor $perms_{flat}(lst)$ generates flat sequences of pairs of call-return events. $perms_{flat}(query_a, query_b)$, *e.g.,* generates

  $$(query_a \ ; \ \underline{reply_a} \ ; \ query_b \ ; \ \underline{reply_b}) \ \Box \ (query_b \ ; \ \underline{reply_b} \ ; \ query_a \ ; \ \underline{reply_a})$$

The two permutation constructors have straightforward formal definitions, for instance, in terms of suitable restrictions of the standard permutation function. The resulting permutation defining functions can then be used during pointcut matching to recognize traces of events corresponding to well-balanced permutations.

General sequence operator. The pointcut VPA is constructed bottom-up by starting from nodes representing basic pointcuts that match single execution events and link them through transitions according to the sequencing, repetition and choices used in a pointcut expression. That is, depending on the type of aspect composition operator used in the pointcut expression, we apply appropriate VPA operations such as adjunction of paths (for choice operator) and concatenation (for sequencing and repetition operators) to build the corresponding composed VPA.

The general sequence operator $;_{\mathcal{I}}$ can be formally defined as follows:

- Modify the matching semantics of the pointcut VPA, so that no interleaving of events is allowed between two consecutive matched events, *i.e.,* that transitions formalize compositions using the sequence operator $;_{\emptyset}$. More formally, if a pointcut p_1 matches the current execution event and if the VPA transition (p_1, p_2) has to be taken next, the next execution event must match p_2. (This is in contrast to the semantics used in [14,4] that allow such interleaving.)

– Define the variants of the general sequence operator that allow interleaving, *i.e.,* $;_{\mathcal{I}}, i \neq \emptyset$, by inserting appropriate paths in the pointcut VPA.

Depth-dependent pointcuts. The pointcut VPA for D_m^n, which represents contexts starting with n open calls of m, can be easily constructed by n transitions representing m that may be interleaved with arbitrarily deep well-balanced contexts involving m and $\underline{m}$. Then the pointcut VPA for the other two depth constructors $D_m^{\leq n}$, $D_m^{>n}$ are built from the VPA for D_m^n based on the following definitions:

$$D_m^{\leq n} := \big\|_{1 \leq i \leq n} D_m^i$$

$$D_m^{>n} := !D_m^{\leq n}$$

Hence, the VPA for $D_m^{\leq n}$ is the result of the concatenation of individual VPA with depth value from 1 to n and the VPA for $D_m^{>n}$ is the complement of that for $D_m^{\leq n}$, which is well-defined because VPA are closed under complement.

Call-closing advice. Advice application formally corresponds to the insertion of transitions corresponding to the advice body into base execution steps. In the small-step semantics, the program then produces the next base execution event and makes the pointcut VPA (and thus the aspect) evolve to the next state. The advice action *closeOpenCall(m, n)* is no different: its effect simply is the insertion of n return operations corresponding to the call m.

4 Preservation of Compositional Properties

In this section we address the problem of whether compositional systems that are subjected to evolution by VPA-based aspects can be proven to preserve fundamental composition properties. Our main point is that, in contrast to general aspect languages such as AspectJ, VPA-based aspect programs are amenable to formal correctness proofs.

We use standard trace-based notions of compatibility and substitutability [23] in this paper. Two protocols are compatible if they do not give rise to any conflict during execution, *i.e.,* no unexpected message is received during collaboration of two components according to their respective protocols. As a simple example consider two nodes that both join the file sharing system and employ two following protocols:

– The first uses the protocol shown in Fig. 3 that allows queries to be aborted when an *abortRequest* is sent in state 2.
– The second uses a protocol that may issue abort requests at any time.

In this case the two protocols are not compatible because the protocol of the first node only allows abort requests to be handled in a specific state not all states.

Substitutability enables a protocol to be used instead of another one in arbitrary contexts. Substitutability of components is typically defined in terms of trace set inclusion: protocol p_1 is substitutable for p_2 if its trace set is a superset of the trace set generated by protocol p_2. For example, assume that p_1 is the protocol that includes nested calls to *query* and *reply* and p_2 is the protocol that consists of only non-nested calls to *query* and *reply*. Then p_1 is substitutable for p_2 since the trace set of p_1 covers that of p_2. Instead of in terms of trace sets only, more precise definitions of both notions can be defined by also taking into account failures over sequences of service requests, see [15].

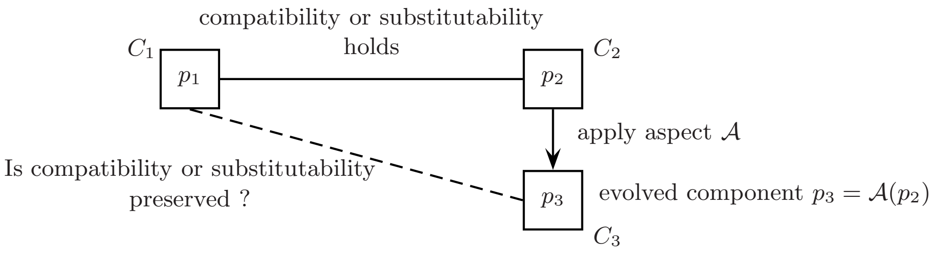

compatibility or substitutability holds
C_1
p_1
p_2
C_2
apply aspect $\mathcal{A}$
Is compatibility or substitutability preserved ?
p_3
evolved component $p_3 = \mathcal{A}(p_2)$
C_3

Fig. 5. Checking for preservation of compatibility/substitutability

Figure 5 illustrates the underlying model of component evolution and the compositional properties we consider. Starting from two protocols p_1, p_2 that constrain the interactions of two collaborating components C_1, C_2 a VPA-based aspect $\mathcal{A}$ is applied to p_2 yielding the protocol p_3 that defines the interactions of the component C_3 after evolution. As indicated in the figure we are interested in two fundamental correctness properties for components, compatibility and substitutability (see, *e.g.,* [15]).

Generally, *e.g.,* if Turing-complete pointcut and advice languages are used for component evolution (as in AspectJ where arbitrary Java methods may be called in `if`-pointcuts and advice), such component properties cannot be proven formally. Furthermore, even in specific cases where a proof is possible, it can typically be performed only in terms of the woven program and not simply in terms of the aspects themselves. VPA-based aspects, however, support formal proofs of such properties because of their limited expressiveness and allow some important properties be proven simply by considering properties of the aspect language only. To this end we propose to exploit the "domain specific" characteristics of VPA-based aspects: proofs over nested contexts as well as regular structures can be performed directly in terms of corresponding features of our pointcut (indexed calls) and advice language (*closeOpenCall*).

Concretely, we demonstrate in the following three different types of proofs of property preservation that are supported by VPA-based aspects:

P1) Proofs that depend only on the properties of the aspect language, *i.e.,* that can be performed in terms of the evolution aspect $\mathcal{A}$ only.

P2) Proofs that can be performed in terms of $\mathcal{A}$ and properties of *classes* of protocols to which p_1 and p_2 belong.
P3) Proofs that require full knowledge of $\mathcal{A}$ and p_1–p_2.

In the following we will present three examples that illustrate the different proof types introduced above.

P1: supporting evolution of error handling. VPA-based aspects are unique (in particular compared to finite-state based approaches) in being able to handle a large class of traversals of distributed recursive algorithms, such as P2P algorithms. Frequently, error handling in such algorithms consists in terminating the exploration of some part of the network and search elsewhere. The action $closeOpenCall(m, n)$ that we have introduced in the advice language directly supports such error strategies by allowing to close n nested calls of the method m.

We can exploit the precisely defined semantics and limited effect of the action *closeOpenCall* to prove some corresponding properties simply in terms of its definition. For example:

> If p_1,p_2 are protocols that recurse using m, p_2 is substitutable for p_1 and aspect $\mathcal{A}$ employs *closeOpenCall* to add returns of m at the end of the execution of protocol p_2, then the adapted protocol p_3 is substitutable for p_1.

Imagine that p_1 is the protocol consisting of recursive calls to *query* and p_2 is an extension of p_1 but explicitly requires one *reply* in its definition (*i.e.,* there can be more than one query but only one reply for the protocol to be complete). Since the trace set generated by p_2 is the superset of that of p_1, p_2 is substitutable for p_1 according to the definition of substitutability. Now assume that protocol p_2 evolves through an aspect that uses the *closeOpenCall* to add a number of *reply* to close all open *query* so that p_2 can work well with another protocol which requires an equal number of replies to the number of queries should be available. The new protocol p_3 resulting from that aspect-based evolution is still substitutable for p_1 since the trace set of p_1 is included in the trace set of p_3.

P2: proving compatibility for depth-cutting heuristics. Recursive distributed algorithms frequently do not unconditionally stop traversals at the top level, but typically do so only in specific contexts. A common example are heuristics that are formulated in terms of the traversal depth from the node where the search has been initiated. Since VPA-based aspects allow the explicit definition of aspects in terms of the nesting depth using the pointcut operator $D^{>k}_{m_c}$, corresponding compositional properties can be proven in terms of properties of this operator and classes of protocols to which it is applied. For example:

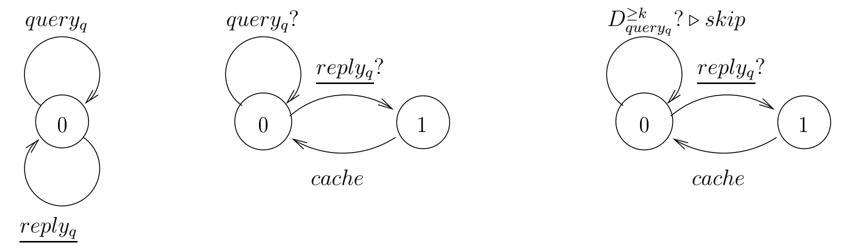
> If
> - p_1 belongs to the class of recursive protocols that repeatedly allows recursive remote calls and returns in m: $\mu a.m_c \,\Box\, \underline{m_c} \;;\; a$,
> - p_2 belongs to the class of protocols that include a remote call to m,

– p_1, p_2 are protocol compatible and
– aspect $\mathcal{A}$ employs a depth-defining operator $D^{>k}_{m_c}$ applying over p_2

Then p_1 and $p_3 = \mathcal{A}(p_2)$ are also compatible.

This property holds because the aspect may only cut calls to m from traces of p_2: the resulting traces remain compatible with those admitted by p_1.

Hence, knowing the specific classes of protocols of p_1, p_2 and class of the aspect modifying them, one can use the above rule to conclude that compatibility is preserved. Let us consider a more intuitive example as illustrated in Figure 6. Fig. 6(a) shows protocol p_1 which provides basic implementation for a recursive query including sequences of queries and replies. Fig. 6(b) illustrates the protocol p_2 that implements an advanced recursive query which allows to cache results. (Note that we use the '?' character to denote remote calls to services provided by another component). According to the definition of protocol compatibility, p_1 and p_2 are compatible. To adapt protocol p_2, an aspect is employed to dynamically skip the calls to *query* for depths greater than k, which is shown in Fig.6(c). We can prove that in this context, p_1 and p_3 are protocol compatible regardless of the change caused by the aspect.



(a) Protocol p_1: basic query

(b) Protocol p_2: extended query with cache

(c) Protocol p_3: adapted protocol of p_2 by aspect

Fig. 6. Aspect-based evolution implemented by depth-defining operator

P3: proving substitutability in terms of p_1,p_2 *and* $\mathcal{A}$. In this example, the classes of protocols and aspect are not sufficient to prove the preservation of compatibility and we need more information on the real values of p_1,p_2 to prove component properties.

Let us reconsider protocols p_1,p_2 as in the first example, *i.e.,* p_1,p_2 involve recursive calls to *query* and p_2 is substitutable for p_1. Assume that now we would like to adapt protocol p_2 in order to cut the depth of queries to k using an aspect with a depth-defining operator $D^{>k}_{m_c}$. In this case the resulting protocol p_3 is in general not substitutable for p_1, since p_1 may admit calls of depth deeper than k. By an analysis of p_1, we may find that the depth limit of p_1 is q and $q \leq k$: we can then prove that p_3 is actually substitutable for p_1.

5 Related Work

There is few related work on aspect-based evolution for component-based systems that considers the preservation of correctness properties for those systems after being changed by aspects. As to the best of our knowledge, this approach is the first exploiting formal methods to investigate the preservation of compositional properties such as compatibility and substitutability for component-based systems that are subject to evolution by protocol-modifying aspects. However, our work still shares common interests with a large body of work covering aspects, components, and applications of formal methods on analysis and verification.

There are some approaches which consider aspect languages that support protocols, most notably [1,5,22]. Approaches [1,5] feature regular aspect languages and a framework for static analysis of interaction properties. The language introduced by Walker and Viggers[22], one of the very few approaches providing non-regular (but not Turing-complete) pointcut languages, proposes tracecuts which provide a context-free pointcut language. However, all of the above approaches do not use the language for an integration of aspects and components or explore the problem of property-preserving for systems that have protocols being modified by aspects. Farías [9] has proposed a regular aspect language for components that admits advice modifying the static structure of protocols and considered proof techniques for the resulting finite-state based aspects.

There exist a large number of proposals that aim at applying AOP over component-based systems, *e.g.,* [10,16,20]. However, the aspect languages in those approaches do not provide explicit support for component protocols. Some of these approaches consider component compatibility, however, in a limited sense to our work: aspects are usually employed in such work to transparently introduce adaptation to components and thus preserve component compatibility. Our approach, in contrast, focuses on preserving protocol compatibility even if aspects have visible effects on interaction protocols.

Few work on evolution of component protocols seems relevant to our work. Braccialia et al. [3] present a formal methodology for automatically adapting components with mismatching interacting behaviors *i.e.,* conflicts at the protocol level. Protocols considered there are expressed by using a subset of $\mu-$calculus. They do not consider how component properties can be proved in terms of proof methods that exploit properties of modification operators. Ryan and Wolf [18] investigate how applications can accommodate protocol evolution. However, this approach concerns mainly syntactic changes on protocols.

Another category of related work is the application of formal methods to analyse aspect systems, such as [11,12]. Our approach differs from those approaches in that we exploit the protocol-based specificities of our aspect language to prove composition properties of software components.

6 Conclusion

In this paper we have motivated the use of aspects to define the evolution of components with explicit interaction protocols. We have motivated and introduced four extensions to our previously defined VPA-based aspect language that are useful in the context of component evolution for distributed applications: a depth-dependent operator, a general sequencing operator, two permutation operator for well-balanced contexts, and an advice operator to close open calling contexts. The first three extensions improve the expressiveness of the pointcut language and make it possible to express common evolution aspects more precisely. The last extension enables aspects to modify interaction protocols in a limited manner, *e.g.,* to support error correction strategies.

Furthermore, we have addressed the problem of preserving compositional properties such as compatibility and substitutability for components that are subject to aspect-based evolution. The main innovation in our approach to handle this problem consists in the exploitation of the aspect language features to reason about properties of components modified by aspects. Concretely, we have shown that our VPA-based aspect language of limited expressiveness admit formal proofs of fundamental compositional properties in the presence of aspect-based evolutions directly in terms of the aspect languages.

With respect to future work, we plan to advance in two directions: language design and proof techniques for property preservation. We strive at the definition of a larger set of VPA-based pointcut constructors for distributed components; furthermore a more powerful advice language for protocol modifications should be included. Moreover, property analysis should take into account modifications made by aspect advice.

References

1. Allan, C., Avgustinov, P., Christensen, A.S., et al.: Adding trace matching with free variables to AspectJ. In: Gabriel, R.P. (ed.) ACM Conference on Object-Oriented Programming, Systems and Languages (OOPSLA), ACM Press, New York (2005)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC 2004. Proceedings of the thirty-sixth annual ACM Symposium on Theory of Computing, June 13–15, 2004, pp. 202–211. ACM Press, New York (2004)
3. Braccialia, A., Brogi, A., Canal, C.: A formal approach to component adaptation. Journal of Systems and Software (2005)
4. Douence, R., Fradet, P., Südholt, M.: A framework for the detection and resolution of aspect interactions. In: Batory, D., Consel, C., Taha, W. (eds.) GPCE 2002. LNCS, vol. 2487, pp. 173–188. Springer, Heidelberg (2002)
5. Douence, R., Fradet, P., Südholt, M.: Composition, reuse and interaction analysis of stateful aspects. In: AOSD 2004. Proc. of 3rd International Conference on Aspect-Oriented Software Development, pp. 141–150. ACM Press, New York (2004)
6. Douence, R., Fradet, P., Südholt, M.: Trace-based aspects. In: Akşit, M., Clarke, S., Elrad, T., Filman, R.E. (eds.) Aspect-Oriented Software Development, Addison-Wesley Professional, Reading (2004)
7. Farías, A.: Un modèle de composants avec des protocoles explicites. PhD thesis, École des Mines de Nantes/Université de Nantes (December 2003)
8. Farías, A., Südholt, M.: On components with explicit protocols satisfying a notion of correctness by construction. In: Meersman, R., Tari, Z., et al. (eds.) CoopIS 2002, DOA 2002, and ODBASE 2002. LNCS, vol. 2519, pp. 995–1006. Springer, Heidelberg (2002)

9. Farías, A., Südholt, M.: Integrating protocol aspects with software components to address dependability concerns. Technical Report 04/6/INFO, École des Mines de Nantes (November 2004)
10. Göbel, S., Pohl, C., Röttger, S., Zschaler, S.: The COMQUAD component model — enabling dynamic selection of implementations by weaving non-functional aspects. In: Proceedings of AOSD 2004, ACM Press, New York (2004)
11. Katz, S., Sihman, M.: Aspect validation using model checking. Verification: Theory and Practice, 373–394 (2003)
12. Krishnamurthi, S., Fisler, K.: Foundations of incremental aspect model-checking. ACM Trans. Softw. Eng. Methodol. 16(2), 7 (2007)
13. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: ICS, pp. 84–95 (2002)
14. Nguyen, D.H., Südholt, M.: VPA-based aspects: better support for AOP over protocols. In: SEFM 2006. 4th IEEE International Conference on Software Engineering and Formal Methods, IEEE Press, Los Alamitos (2006)
15. Nierstrasz, O.: Regular types for active objects. In: Nierstrasz, O., Tsichritzis, D. (eds.) Object-Oriented Software Composition, ch. 4, pp. 99–121. Prentice-Hall, Englewood Cliffs (1995)
16. Pessemier, N., Seinturier, L., Coupaye, T., Duchien, L.: A model for developing component-based and aspect-oriented systems. In: Löwe, W., Südholt, M. (eds.) SC 2006. LNCS, vol. 4089, pp. 259–273. Springer, Heidelberg (2006)
17. Plasil, F., Visnovsky, S.: Behavior protocols for software components. IEEE Transactions on Software Engineering 28(9) (January 2002)
18. Ryan, N.D., Wolf, A.L.: Using event-based translation to support dynamic protocol evolution. In: ICSE 2004. Proceedings of the 26th International Conference on Software Engineering, pp. 408–417. IEEE Computer Society, Washington, DC, USA (2004)
19. Südholt, M.: A model of components with non-regular protocols. In: Gschwind, T., Aßmann, U., Nierstrasz, O. (eds.) SC 2005. LNCS, vol. 3628, Springer, Heidelberg (2005)
20. Suvée, D., Vanderperren, W., Jonckers, V.: JasCo; an aspect-oriented approach tailored for component-based software development. In: AOSD 2003. Proc. of 2nd International Conference on Aspect-Oriented Software Development, pp. 21–29. ACM Press, New York (2003)
21. Vanderperren, M., Suvee, D., Cibran, M.A., De Fraine, B.: Stateful aspects in JAsCo. In: Gschwind, T., Aßmann, U., Nierstrasz, O. (eds.) SC 2005. LNCS, vol. 3628, Springer, Heidelberg (2005)
22. Walker, R.J., Viggers, K.: Implementing protocols via declarative event patterns. In: FSE-12. Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 159–169. ACM Press, New York (2004)
23. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Transactions of Programming Languages and Systems 19(2), 292–333 (1997)

Multi-stage Aspect-Oriented Composition of Component-Based Applications

Bert Lagaisse, Eddy Truyen, and Wouter Joosen

Dept. of Computer Science, K.U.Leuven
Celestijnenlaan 200A, 3001 Heverlee, Belgium
{bertl, eddy, wouter}@cs.kuleuven.be

Abstract. The creation of distributed applications requires sophisticated compositions, as various components — supporting application logic or non-functional requirements — must be assembled and configured in an operational application. Aspect-oriented middleware has contributed to improving the modularization of such complex applications, by supporting a component model that offers aspect-oriented composition alongside the traditional composition of provided and required interfaces. One of the recent advances in AO middleware is the ability to express dynamic compositions that depend on the evaluation of available context information — some of this information may only be available at deployment time.

The search for high level composition mechanisms is an ongoing track in the research community, yet the composition logic of a real world application remains complex and it would greatly pay off if composition logic — traditionally encoded in monolithic deployment descriptors — could be reused over ranges of applications and even be gradually refined for specific applications.

This paper presents M-Stage, an AO component and composition model that supports the reuse and adaptation of compositions in distributed applications that are built on AO middleware. We illustrate the power of M-Stage by applying the model in a realistic distributed application where we analyze the reuse and adaptation potential of the M-Stage model.

1 Introduction

Distributed applications are typically built on middleware that offers a component model and execution environment for these applications. Practical middleware platforms nowadays have to support complex composition of application components and have to support a broad range of services that deal with the non-functional concerns in a distributed application.

Aspect-Oriented middleware (AO middleware, AOM) has contributed to improving the modularization of such complex applications, by supporting an aspect-component model that offers aspect-oriented composition (AO composition) alongside traditional composition of provided and required interfaces [7,8,14,17,18]. The core concept in AO composition is the aspect[3]: a coherent abstraction that encapsulates one specific (often crosscutting) concern in a separate software module. An aspect defines behavior that can be executed and defines composition logic to describe where and when this behavior should be executed. AO middleware typically separates aspect behavior and composition logic from each other. The composition logic is specified externally in a deployment descriptor while the aspect behavior is represented within traditional components as methods. Composition logic in AOM is thus specified in the form of *Whenever event X in the application occurs, execute method behavior Y of component Z*. An example of such composition logic can be: *whenever a component operation is executed, execute the enforcement method of the Authorization component*. One of the recent advances in AO middleware is the ability to express dynamic compositions that depend on the evaluation of available context information about the distributed infrastructure. This context information may include for example the component names of involved components (e.g. caller and callee of an operation), their container, the application they belong too, the hosts they are deployed on, etc. As a result complex composition with remote events can be expressed more concisely and at a higher-level of abstraction. An example of a such powerful composition logic can be *Whenever a client in the ATM-network calls a component on the application server, the Kerberos authentication scheme must be applied*.

The search for high level composition mechanisms is an ongoing track in the research community, yet the composition logic of a real world application remains complex and it would greatly pay off if composition logic — traditionally encoded in monolithic deployment descriptors — could be reused over ranges of applications and even be gradually refined for specific applications. Before elaborating on this problem we will first define the concept of AO composition more rigorously.

Basic concepts of AO Composition. A key element in the specification of the composition logic is the concept of a *pointcut* which is a description of a set of join points where aspects should execute. *Join points* represent dynamic, runtime conditions that arise during program execution. The occurrence of such a condition is an event that can trigger the execution of aspect behavior. *Advice* specifies what aspect behavior should be executed and when the aspect behavior should be executed (typically before, after or around the event) [4]. Pointcuts select join points by declaratively specifying the kind and context of join points. The *kind* of a join point refers to the type of instruction being executed. For example, two different kinds of join points are method call and field access. The *context* of joint points refers to additional information that can be made available to constrain the pointcut such as the method signature, type and identity of the caller or callee of a method, and various distributed infrastructure properties as mentioned above. Which kind of context and which available context that are supported by an AOM, are defined by the AOM's *join point model*. In general, the richer the join point model, the more complex compositions can be supported.

1.1 Problem Statement

In this section, we set up the scene for the rest of the paper. The goal is to identify shortcomings of current AO middleware with respect to supporting the development of reusable composition logic for late integration in various applications and deployment environments. In the following subsections we will illustrate these shortcomings using a pedagogical example in the context of the AO middleware DyMAC [9]. DyMAC has one of the richest join point models and therefore serves as a good example to illustrate these shortcomings. After this illustration we will shortly summarize the general problem statement of this paper.

The thread of the examples in this paper is the construction of a family of banking applications. To illustrate the shortcomings in this section and to illustrate the basic solution in the next section, we use a pedagogical, more concise example in that application domain. For the validation of the solution in section 3 we define a more elaborate case study, based on the example defined in this section.

Problem 1: Reuse of composition logic across applications. Consider the construction of a family of banking applications from a set of components using AO composition and traditional composition. One of these components is the *Authorization* component which verifies the application-level rights of authenticated users when a banking transaction is executed. Another component is the *Account* component which is a generic software module that can be reused for different banking products such as a basic checking account service, for custody accounts in an investment application, or to keep track of loan balances in a loan application. Consider in particular two specific banking products, a *basic banking service* and an *investment application* that use the Account and the Authorization component. Each application has a facade component that uses the Account entity: *BasicBanking* and *Investment*. The interfaces of Account, BasicBanking and Investment are as follows:

```
interface IBasicBanking {
  void CreateAccount(string id, string owner);
  void Deposit(string id, double amount);
  void Withdraw(string id, double amount);
  void Transfer(string from, string to, double amount);
  AccountInfo GetInfo(string id);}
interface IInvestment {
  void CreateAccount(string id, string owner);
  void Deposit(string id, double amount);
  void Withdraw(string id, double amount);
  void BuyStock(string account, string StockId, int amount, double maxprice);
  AccountInfo GetInfo(string id);}
interface IAccount {
  string GetId();
  double GetBalance();
  void Deposit(double amount);
  void Withdraw(double amount);
  List<Transaction> GetTransactions();}
```

As authorization is a crosscutting concern it is composed using AO composition descriptors of DyMAC, which is illustrated in Figure 1. The AO composition of authorization with the basic banking application specifies for example that the authorize method of the Authorization component should be executed around the execution of the Deposit and WithDraw operation of Account, but also around the execution of the Transfer operation of BasicBanking. Figure 1 also shows the AO composition descriptor that composes the Authorization component in the investment application. Notice that large part of this composition logic is the same as in the basic banking application. Yet currently, the developer has no option then to duplicate this common composition logic in two separate descriptors. This is obviously problematic in the presence of maintenance and evolution. A preferable solution is to modularize this common part in a separate AO

```
ao-composition{
 pointcut{
  kind: execution;
  signature: void Deposit(..) OR void WithDraw(..)
     || void Transfer(..);
  Callee{
   Component: Account OR BasicBanking;}
  Caller{
    Host: NOT *.intranet.bank.com}}
 advice{
    advice-component: Authorization;
    advice-method: authorize;}}
```

```
ao-composition{
 pointcut{
  kind: execution;
  signature: void Deposit(..) OR void WithDraw(..)
     || void BuyStock(..);
  Callee{
   Component: Account OR Investment;}
  Caller{}
 advice{
    advice-component: Authorization;
    advice-method: authorize;
 }}
```

Fig. 1. AO composition of Authorization with the basic banking application and the investment application respectively. The underlined specifications are the same in both AO composition descriptors. Deployment-specific composition logic is tangled into the composition logic of basic banking. The specification in bold indicates the evaluation of deployment-specific context info.

composition descriptor, reuse it for a particular application, and non-invasively refine it to needs of the specific application.

Problem 2: Reuse across deployment environments. Similarly, AO composition logic cannot easily be reused across multiple deployment environments. The composition of authorization may also depend on deployment-specific information in order to enforce a deployment-specific policy. For example in Figure 1, a security policy is superimposed that authorization should not be performed when the call originates from the trusted intranet zone. However, this distributed context information is specific to the concrete deployment environment of basic banking. Encoding this deployment-specific composition logic at the same level of abstraction will cause difficulties when trying to reuse the common composition logic for another deployment environment. To support reusability of the common composition logic, it should be possible to non-invasively extend pointcuts with evaluation of deployment-specific context information. Furthermore, if specifying reusable composition logic is the goal, the aspect-component model should enforce the developer to abstract from context information that is specific to a particular application or deployment environment. For example, a component builder who has to create a reusable secure account component (by composing Account and Authorization) should not be permitted to specify pointcuts that evaluate the host name of calling components. This context information should not be made inaccessible for the developer; rather it should be made irrelevant at a certain level of abstraction.

Summary of the problem. One of the recent advances in AO middleware is the ability to express complex AO compositions with remote events. This strength, however, is also a weakness: when common composition logic in a family of applications must be duplicated in separate deployment descriptors, issues of reuse arise. All AO middleware that use a monolithic representation of pointcuts have difficulty defining reusable composition logic because they do not provide modularization and gradual refinement of composition logic. Furthermore, the ability to abstract from context information when possible, but provide access where necessary, is particularly important for AO middleware with a rich join point model. Although a rich join point model allows to express complex composition logic with remote events, the monolithic representation of pointcuts causes the tangling of the evaluation of various context information. Some of this information may only be available during application construction or at deployment time. This causes additional difficulties with reuse of composition logic.

1.2 The M-Stage Solution

This paper presents M-Stage, an aspect-component model that supports the reuse and gradual refinement of AO composition logic on top of AO middleware. An essential element of our solution is that M-Stage integrates AO composition with the hierarchical and phased approach of Component-Based Software Development (CBSD). State-of-the art component models in middleware such as J2EE [23] and CCM [22] typically organize software deployment in multiple subsequent *stages*. These stages include hierarchical component aggregation first, then application assembly and finally application deployment. M-Stage supports specification and gradual refinement of AO composition logic across these multiple stages. We call this *multi-stage composition*. The major contributions of M-Stage are:

- Support for hierarchical modularization of AO composition logic in separate aggregation, assembly and deployment descriptors.
- Support for gradual refinement of AO composition logic across these multiple stages.
- A join point model that abstracts from certain context types during the early stages of aggregation and assembly. These details are thus made irrelevant for the developer at a certain stage to ensure reusability of common composition logic across multiple applications and deployment environments. We call this a multi-stage join point model, which is graphically depicted in Figure 2.

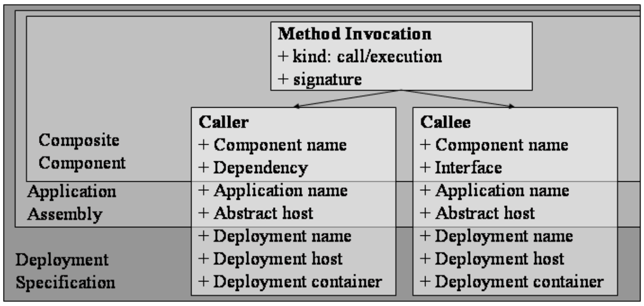


Fig. 2. Multi-stage joinpoint model

The rest of the paper is structured as follows. In section 2 we describe the M-Stage aspect-component model. In section 3 we illustrate M-Stage by means of a case study and evaluate the overall approach. Section 4 summarizes related work. We conclude in section 5.

2 M-Stage

The development of a distributed component-based application involves multiple composition and configuration activities. These activities can be classified in four *stages*:

1. The specification of elementary components, which specifies the provided and required interfaces of elementary components and their implementation.
2. The aggregation of composite components, which consists of repetitive, hierarchic composition of elementary and composite components.
3. Assembling the application, which includes composing the different (aggregate) elements of the application and defining an abstract architecture for it.
4. The deployment specification of the application, which maps the software components to their concrete deployment location.

Each stage leads to specific artifacts, such as elementary components, composite components, application assemblies and deployment specifications. These artifacts are produced by different stakeholders : component providers, application assemblers and application deployers. Composition of components occurs in three stages: hierarchical component aggregation first, then application assembly and finally application deployment. M-Stage supports modular specification and gradual refinement of AO composition logic across these multiple stages. Refinement of an AO composition can occur (repeatedly) within one stage, and also across stages. The M-Stage AO composition model offers a multi-stage joinpoint model, in such way that it supports contextual properties specific to each stage. The model also restricts the visibility of those properties to the appropriate stages.

First, we define the basic component model of M-Stage. We define what kind of artifacts are created in each of the different stages. Second, we define the multi-stage joinpoint model, multi-stage AO composition and the support for composition refinement.

2.1 The M-Stage Component Model

We define the basic component model with the four different stages and we explain which information is specified in the different kinds of artifacts in each stage.

Elementary Components. The elementary components are object-based entities with well-defined interfaces. An elementary component can have multiple interfaces. It can also have a set of dependencies that specify the required interfaces of the component. The specification (or descriptor) of an elementary component defines (1) the component name, (2) the provided interfaces, (3)the implementation file and (4) the dependencies of the component. A dependency is defined by a dependency name and an interface that is expected to be bound to the dependency.

As an example we describe the Account component in the next listing. It provides the IAccount interface as defined in section 2. It has a dependency *tx* for the required ITransaction interface. The implementation is defined in AccountImpl. The numbered labels in comment refer to the enumeration of the elements as introduced in the previous paragraph.

Composite components. A composite component is a hierarchical composition of a set of components. The components that are part of the composite component can be elementary or composite components. The specification of a composite component defines (1) a name for the composite, (2) the set of components it contains, and (3) composition specifications. These compositions can be normal dependency - component mappings or AO compositions. AO compositions are further discussed in detail in the next subsection.

An example of a composite component is illustrated in the next listing. We define the bank's generic account component: *SecureAccount*, an aggregation of *Account*, *Transaction* and *Authorization*. We define the dependency-component mapping of Account with Transaction. The AO composition of Authorization in this composite is discussed later.

```
component Account { //1
  provides: IAccount; //2
  implementation: AccountImpl; //3
  dependency tx: ITransaction; //4
}
```

```
composite SecureAccount {
  contains: Account, Transaction,
    Authorization;
  composition{
    dependency: Account.tx;
    component: Transaction;}
  ao—composition {...}}
```

Application assemblies. Application assemblies define distributed component-based applications that are deployable to multiple environments. They consist of a set of components (elementary or composite), similar to composite components. The assembly specification contains reusable architectural information about the application by specifying an abstract deployment topology. This is a set of abstract hosts on which the components are allocated. A possible architecture that could be included is a multi-tier architecture or a simple client-server architecture. Concretely, the specification of an application assembly contains (1) a name for the application, (2) the set of components it contains, (3) definitions of abstracts hosts, (4) mappings of components to abstract hosts and (5) composition specifications specific to the application.

The example in Figure 3 illustrates an abstract architecture for a banking application: SecureAccount and BasicBanking are located on the abstract appserver, EmployeeClient on the workstations and ATMClient on the ATM machines.

Deployment specification. A deployment specification of an application specifies (1) the application it deploys, (2) unique deployment names for the public accessible components, (3) mappings of abstract hosts to concrete hosts, (4) mappings of a component to a container on a concrete host, (5) a set of already deployed components it uses and (6) compositions specific to the deployment environment.

The deployment example in Figure 3 deploys the banking application defined above. It maps the abstract host *appserver* to the concrete host *appserver1.mybank.net*, and deploys the BasicBanking component with the unique name MyBasicBanking on a concrete container on the application server. It also declares that it uses an already deployed component in the environment : runtime monitor, which will monitor and log the distributed execution trace of all sessions on the application server's containers. We illustrate the deployment-specific compositions further on.

```
application BasicbankApp { //(1)
 contains: SecureAccount, BasicBanking,
        EmployeeClient, ATMClient; //(2)
 abstracthost: atm, workstation,
        appserver; //(3)
 locate{ //(4)
  component: BasicBanking, SecureAccount;
  abstracthost: appserver;}
 locate{
  component: EmployeeClient;
  abstracthost: workstation;}
  ...
  //(5): compositions
}
```

```
deployment MyBank{
 contains: BasicbankApp; //(1)
 map{ //(3)
  abstracthost: BasicbankApp.appserver;
  host: appserver1.mybank.net;}
 deploy{ //(4)
  component:BasicbankApp.BasicBanking;
  deploymentname: MyBasicBanking; //2
  host: appserver1.mybank.net; //4
  container: container1;}
 usedeployed: RuntimeMonitor; //(5)
 //(6) compositions
}
```

Fig. 3. Application assembly and deployment specification

2.2 AO Composition Model

Composition of components is supported in three stages: (a) composite component aggregation, (b) application assembly and (c) deployment. We continue the description of the composition model by focusing on AO composition. We first explain the multi-stage joinpoint-model. Then we elaborate on the multi-stage AO compositions with support for composition refinement.

Multi-stage joinpoint model. The joinpoint model defines the kind and context of joinpoints. The kind of the joinpoint can be either a call or an execution of a method. The contextual information available about the calling and called component depends on the stage in which the AO composition is defined. For each stage we define a set of contextual properties in the joinpoint model that can be evaluated in that specific stage. The joinpoint model enforces that a stakeholder specifying a certain AO composition in a certain stage only has to reason over the contextual information that is relevant at that stage. A definition of the contextual properties in each stage is explained next, and is also depicted in Figure 2.

In the composite aggregation stage, the contextual properties about the calling component are the component name and the dependency that is used. Contextual properties about the called component are the component name and the interface on which the method invocation is done. In the application assembly stage, the application name and abstract host of caller and callee become available, next to the already available properties of the composite aggregation station. The contextual properties that become available in the deployment stage are the component's deployment name, host, and container.

Multi-stage AO composition. AO compositions can be specified in each stage that supports composition of components. Such an AO composition consists of three parts: (c1) a name for the AO composition, (c2) a pointcut expression and (c3) a set of advices. Figure 4 shows the high-level grammar, which is the same for all stages. We discuss the details of a pointcut expression and an advice next.

A pointcut expression evaluates over the kind and context of the joinpoints. The kind of the joinpoint can be either a call or an execution of a method invocation. Pointcuts can further evaluate over the contextual properties of the joinpoint. As defined in the

```
ao−composition <name>{ //(c1)
  Pointcut <name>{ //(c2),(p1)
    Kind: [call|execution]; //(p2)
    Signature: <method−pattern>; //(p3)
    Caller{
      [<property>: <string−pattern>;]*} //(p4)
    Callee{
      [<property>: <string−pattern>;]*}} //(p5)
  [Advice <name>{ //(c3)
    Advice−Component: <component−name>;
    Advice−Type: [before|after|around];
    Advice−Method: <interface>.<advice−method>;
  }]*
}
```

Fig. 4. Grammar for AO compositions

grammar, a pointcut expression consists of (p1) a name for the pointcut and evaluates over (p2) the kind of joinpoint, (p3) the method signature, (p4) caller properties (contextual properties about calling component) and (p5) callee properties (contextual properties about the called component). The available contextual properties about caller and callee depend on the stage in which the pointcut is specified. Furthermore, if pointcuts do not specify a value for a certain property, it has a default value. This default value is the least restricting value for that contextual property. Furthermore, an advice is described by (1) the component that provides the aspect behavior, (2) an advice method of the advising component and (3) an advice type (before, after or around).

A first example is SecureAccount's AO composition, specified in the aggregation stage. It illustrates the use of joinpoint properties that are only visible in the aggregation stage. A second example is RuntimeMonitor's AO composition, defined in the deployment stage. It illustrates the modularization of deployment specific composition logic: the pointcut as well as the advice are deployment-specific.

```
composite SecureAccount{
 contains: Account, Transaction,
   Authorization;
 ...
 ao−composition aoc1{
  Pointcut sensitive{
   Kind: execution
   Signature: void Deposit(..)
     OR void Withdraw(..);
   Callee{
    Component: Account;}}
  //advices of Authorization to call.
  Advice checkrole{
    Advice−Component: Authorization;
    Advice−Method:
      IAuthorization.VerifyRole;
    ...
 }}}
```

```
deployment MyBank{
 ...
 ao−composition monitor{
  Pointcut monitortrace{
   Kind: execution
   Signature: * *(..);
   Callee{
    Host: appserver1.mybank.net;}}
  Advice logtrace{
   Advice−Component: RuntimeMonitor;
     ...
  }}
```

Refining AO compositions. When a composition artifact is used in an other artifact, of possibly another stage, it might be necessary to further refine the pointcuts. This refinement can be a different evaluation of an existing contextual property or the use of a newly available property in the actual stage. The *with* directive allows to refine the pointcut in an AO composition or in parts of it. The *old* keyword allows to refer to the previous pointcut definition. The refined pointcut is expressed by referring to *the pointcut name within its naming path*. First we define the structure of the naming path, then we define the specification of a refinement.

```
<namingpath> =
<artifact >[.<containingartifact >]*.<ao−composition >
```

```
<artifacttype > <name> {
   ...
   with <namingpath >.<pointcut> {
     Signature = <new value >;
     with Caller {
       //refinement of properties
      }
     with Callee{
      //refinement of properties
     }}}
```

As an illustration of reuse and refinement we reconsider the examples from the introduction. SecureAccount's AO composition of Account and Authorization has already been defined earlier. The stepwise refinement of the AO composition for the basic banking application is defined in the Figure 5. The AO composition of authorization is refined in the definition of the basic banking application assembly, so that it is applied to BasicBanking too, and not applied to Account when the calls come from BasicBanking. In the deployment specification of the basic banking application, the authorization is refined so that it is never enforced when the calls originate from the trusted intranet zone.

```
Application BasicbankingApp{
  contains: SecureAccount;
  contains: BasicBanking;
  with SecureAccount.aoc1.sensitive{
    Signature: old.Signature || void Transfer(..)
    with Callee{
      Component: old.Callee.Component || BasicBanking;}
    with Caller{
      Component: !BasicBanking;}}}}
```

```
deployment MyBank{
 ...
 with BasicBankingApp.SecureAccount.aoc1.sensitive{
   with Caller{
     Host: !*.intranet.bank.com;}}}}
```

Fig. 5. Refinement of Authorization

3 Validation

In this section, we validate the power of M-Stage by applying the model in a family of e-finance applications. Figure 6 depicts the core assets of this family, which are (1) a set of reusable elementary components, which can be reused for different e-finance applications, (2) a set of reusable composite components encoding reusable composition

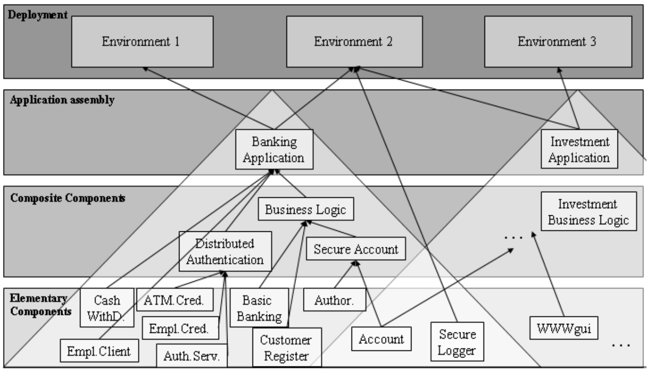


Fig. 6. Multi-stage AO composition

logic, and (3) reusable application assemblies encoding reusable architectures that are deployable to multiple deployment environments.

We illustrate how M-Stage[1] supports this and how a concrete family instance is built, using multi-stage composition and gradual refinement. We evaluate against the reuse potential of DyMAC's own component model.

3.1 The Elementary Components

Three elementary components are typical business components: *CustomerRegister*, *BasicBanking* and *Account*. The CustomerRegister and BasicBanking components are remotely accessible services that offer the core business operations to manage customers, create new accounts and execute transactions. The BasicBanking component uses the Account component, which is an entity that contains the basic information about accounts: a unique account id, a balance, and a record of transactions executed on the account. The account component is a generic account that offers two operations: deposit and withdraw. The interfaces of the three core business components are defined in Figure 7.

The employees at the branch offices of the bank use the *EmployeeClient* component at their workstations to manage the customers' accounts and to handle customer requests. The customers can also use an ATM to withdraw money from their accounts. The *CashWithdrawal* component on the ATM terminals as well as the EmployeeClient component uses the BasicBanking component to execute the transactions. The different components in the application are depicted in Figure 8. Figure 8 also describes the set of elementary aspect-components.

There are four collaborating aspect-components for authentication in the application. First, a client-side component, *EmployeeCredentials*, asks the employees for credentials, before the EmployeeClient component starts up. A second component, which is a local component on the ATM, *ATMCredentials*, asks the ATM-users for credentials

[1] We use the M-Stage component model implementation on top of the DyMAC runtime environment.

```
interface IBasicBanking {
  void CreateAccount(string id, string owner);
  void Deposit(string id, double amount);
  void Withdraw(string id, double amount);
  void Transfer(string from, string to, double amount);
  AccountInfo GetInfo(string id);}
interface ICustomerRegister {
  void CreateCustomer(string id, string name,...);}
interface IAccount {
  string GetId();
  double GetBalance();
  void Deposit(double amount);
  void Withdraw(double amount);
  List<Transaction> GetTransactions();}
```

Fig. 7. Interfaces of BasicBanking, CustomerRegister and Account

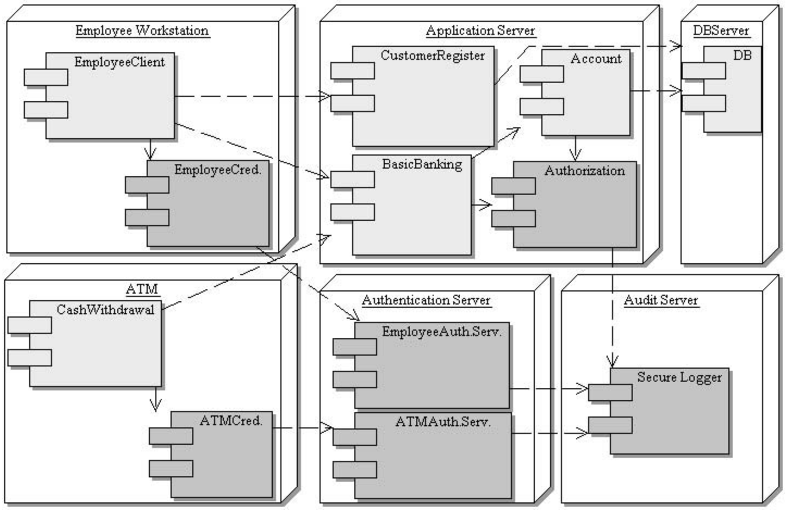


Fig. 8. Application overview

when they want to withdraw cash. Third, a server-side component *EmployeeAuthenticationService* authenticates the credentials of the employees and generates an employee authentication token. Forth, the *ATMAuthenticationService* authenticates the credentials of atm users and generates an atm authentication token. Both authentication services are located at the central authentication server and are called after the client has provided his credentials. The returned authentication token is stored in the client side credential components. Each time a call is made from the clients to the application server, an advice in the client side credential components will push the stored authentication token along with the call. A fifth aspect-component is the *Authorization* component that verifies the application-level rights associated with the authenticated user when a banking transaction is executed. Employees can only do transactions during office hours. ATM-users can only withdraw from their own account, with a maximum of 5000 Euro. The sixth aspect-component is the *SecureLogger* component at the central audit server. It keeps track of all authentication and authorization attempts and of the results.

The interfaces of the advising components in the example are defined as follows. EmployeeCredentials and CustomerCredentials provide the interface ICredentials. It defines an operation to get the credentials of the user, an operation to store the authentication token after it is returned by the authentication service and an operation to push the authentication token with every call that is made to the application server.

```
interface ICredentials{
  void GetCredentials(RuntimeJoinPoint rjp);
  void StoreToken(RuntimeJoinPoint rjp);
  void PushToken(RuntimeJoinPoint rjp);}
interface IAtmAuthenticationService{
  void VerifyAtmCredentials(RuntimeJoinPoint rjp);}
interface IEmployeeAuthenticationService{
  void VerifyEmployeeCredentials(RuntimeJoinPoint rjp);}
interface IAuthorization{
  void VerifyRole(RuntimeJoinPoint rjp);
  void VerifyTime(RuntimeJoinPoint rjp);
  void VerifyOwner(RuntimeJoinPoint rjp);
  void VerifyAmount(RuntimeJoinPoint rjp);
interface ISecureLogger{
  void Log(RuntimeJoinPoint rjp);}
```

3.2 Composition and Deployment of the Basic Banking Application

The composition of the basic banking application, as depicted in Figure 6, defines 11 AO compositions (9 definitions + 2 refinements) across the composite components (5+1), the application assembly (2+0) as well as the deployment specification (2+1).

Composite components. The composite components in the systems are SecureAccount, BusinessLogic and DistributedAuthentication. The composite component *SecureAccount* contains Authorization and Account. It specifies one AO composition enforcing the authorization rules on the transaction operations of Account.

```
composite SecureAccount{
 contains: Account, Authorization;
 ao-composition aoc1{
  Pointcut sensitive{
   Kind: execution
   Signature: void Deposit(..) || void Withdraw(..);
   Callee{
    Component: Account;}}
  //advices of Authorization to call.
  Advice checkrole{
    Advice-Component: Authorization;
    Advice-Type: before;
    Advice-Method: IAuthorization.VerifyRole;}
  ... }}
```

BusinessLogic is a composite component containing SecureAccount and BasicBanking. When the SecureAccount composite is reused in the BusinessLogic composite, the AO composition is refined to also enforce authorization on the basic banking service. Security checks need to be verified as early as possible in the call-chain. Calls coming from BasicBanking to SecureAccount then need no authorization. Concretely, the

Signature property will be broadened with the Transfer method, and the component name of the callee will be broadened with the BasicBanking component name. The component name of the caller will be restricted from its default value (*) to !BasicBanking because authorization would not occur twice.

```
composite BusinessLogic{
  contains: SecureAccount;
  contains: BasicBanking;
  with SecureAccount.aoc1.sensitive{
    Signature: old.Signature || void Transfer(..)
    with Callee{
      Component: old.Callee.Component || BasicBanking;}
    with Caller{
      Component: !BasicBanking;}}}}
```

DistributedAuthentication is a composite component containing the authentication components in the application. This component encapsulates four AO compositions specifying the following internal interactions between the authentication components. First, after gathering the credentials of the employee, the employee authentication service is called to verify the credentials and return an employee authentication token. Second, after the authentication token is returned it is stored in the employee credential component at the client side. In the next listing, we define these two AO compositions of EmployeeCredentials and EmployeeAuthentication in DistributedAuthentication. The other two AO compositions for ATM authentication are similar.

```
composite DistributedAuthentication{
 contains: EmployeeCredentials, EmployeeAuthenticationService ,...;
 ao-composition checkEmployee{
  Pointcut credentials{
   Kind: execution
   Signature: * GetCredentials(..);
   Callee{
    Component: EmployeeCredentials;}}
  Advice {
    Advice-Component: EmployeeAuthenticationService;
    Advice-Type: after;
    Advice-Method: IEmployeeAuthenticationService.VerifyCredentials;}}
 ao-composition storeEmployeeToken{
  Pointcut verify{
   Kind: call;
   Signature: * VerifyCredentials(..);}
  Advice storeToken{
   Advice-Component: EmployeeCredentials
   Advice-Type: after;
   Advice-Method: IEmployeeCredentials.StoreToken;}}
 ao-composition checkAtmUser {...}
 ao-composition storeAtmToken {...}}
```

Application Assembly. The application *BankingApplication* assembles 4 components: BusinessLogic, CashWithdrawal, EmployeeClient and DistributedAuthentication. The two AO compositions between DistributedAuthentication and the other application components are defined in the application assembly. Concretely, the two advices that get the credentials, defined in ATMCredentials and EmployeeCredentials, are composed as before advice on the entry-methods of the client components (e.g. the main method of

EmployeeClient). We define the composition in BankingApplication of DistributedAuthentication with the employee client and BusinessLogic: the employee authentication token is pushed along with all calls leaving a workstation towards the appserver. The other AO composition to push the atm token is similar.

```
application BankingApplication{
 contains: DistributedAuthentication ,...
        EmployeeClient , ATMClient;
 abstracthost: atm, workstation ,
        appserver;
 locate{
  component: BusinessLogic;
  abstracthost: appserver;}
 locate{
  component: EmployeeClient;
  abstracthost: workstation;}
 ao−composition pushEmployeeToken{
  Pointcut employeeCalls{
   Kind: call
   Signature: void *(..);
   Caller{
    abstracthost: workstation;}
   Callee{
    abstracthost: appserver;}}
  //Advice push employee token ...
 }
 ao−composition pushATMToken{...}}
```

```
deployment MyBank{
 ...
 with BankingApplication.BusinessLogic
      .SecureAccount.aoc1.sensitive{
   with Caller{
     Host: !applicationserver;}}
 ao−composition authentication_audit{
  Pointcut operationstoaudit{
   Kind: execution
   Signature: * *(..);
   Callee{
    Host: authenticationserver;}}
  //Advice: log after exception
 }
 ao−composition authorization_audit{
}
```

Deployment specification. In the deployment specification above, the secure logger at the audit server of a specific deployment environment is composed with the authentication services and with the authorization component. A failed credential verification or authorization will be recorded in the audit trail by the secure logger. The host name of the authentication server is used in the pointcut in stead of enumerating all authentication services. Failures of any authentication service on the host will then be logged.

The refinement of the authorization pointcut in the deployment specification is a refinement based on new contextual information: all components in BusinessLogic are deployed on the same container on the applicationserver. Therefor, if an invocation comes from within the applicationserver, authorization should not be applied. The host property of the caller will therefore be restricted from its default value * to !applicationserver.

3.3 Evaluation

The goal of this section is to quantify to which extent M-Stage improves reusability of AO composition logic. To achieve this goal we have measured how many AO compositions in the above banking application can be reused in another hypothetical usage context (for example, the investment application) and we have compared this with composition descriptors in DyMAC. We have used the following metrics in particular:

A The total number of AO compositions defined.
B The number of lines of code defined.
C The number of AO compositions that can be reused across multiple applications.
D The number of AO compositions that can be reused across multiple deployment environments.

The table below gives an overview of our results of this comparison. For this relatively small configuration (83 LOC), M-Stage allows to reuse 55% of the AO composition logic across applications, and 72% across deployment environments. This gain of reuse comes at the cost of 22% more AO compositions that must be specified due to the gradual refinement, and an increase of 18% in total lines of code. We have achieved the increase of reuse because of four main points:

1. Pointcuts in the application assembly per definition do not contain deployment host information and therefore don't tie the application to a fixed deployment environment.
2. Deployment-time AO compositions do not need to be defined in the application assembly.
3. Reusable AO compositions that should already be defined in a reusable composite component can be localized in a separate descriptor.
4. It is possible to refine existing AO compositions.

with refinement necessary	**A**	**B**	**C**	**D**
DyMAC	9	83	n/a	n/a
M-Stage	11 (9+2)	98	6 (5+1) out of 11	8 (7+1) out of 11
Relative increase	+22%	+18%	n/a	n/a

4 Related Work

Three categories of related work are considered: AO middleware, stepwise refinement models and model-driven middleware.

AO middleware (AOM). The general relation between M-Stage and AOM platforms has already been discussed in the motivation of this paper. Summarized, M-Stage augments existing AOMs with *multi-stage composition*, which modularizes AO compositions across multiple stages, and with a *multi-stage join point model*, which enables abstraction of certain context information during the early stages. Multi-stage composition pays off for all AOMs, whereas the multi-stage join point model only pays off for AOMs with a rich join point model. For example, JBoss AOP [8], AspectWerkz [15], Spring AOP [14], Prose [17], CAM/DAOP [10], DADO [18], FAC [28] and GlueQoS [19] only benefit from multi-stage composition because these platforms do not support the evaluation of context properties concerning the application architecture or deployment. On the other hand platforms such as JAC [7], DJCutter [20], AWED [11] and DyMAC [9] also benefit from a multi-stage joinpoint model because they do support the evaluation of distributed context and application architecture.

Stepwise refinement. Batory et al. presents a software composition model and associated tool set, called AHEAD [16], that supports large-scale refinement of aspect-like modules in a product family. Atkinson and Kühne present the concept of stratified architectures [21] for gradually refining and introducing aspect behavior across multiple levels of abstraction in the design of a distributed application. There are two important differences between these works and M-Stage. First M-Stage has a more focused goal: it supports stepwise refinement of aspect composition logic, not aspect behavior. Furthermore, M-Stage supports stepwise refinement across multiple development stages and not across multiple levels of abstraction in the design of a software system. Finally, the AHEAD tool set does not target AO middleware. Having said this, it should be noted that AHEAD supports a uniform *compose* operation that also targets *non-code artifacts*. As such it is possible that AHEAD can also be used to express stepwise refinement of composition descriptors. Exploring this interesting idea is however subject to further research.

ADL-driven and Model-driven middleware. CAM/DAOP [10] is an AO middleware that specifies the composition of components and aspects using an architectural description language (ADL) [29], named DAOP-ADL [27]. This ADL-based approach provides an interesting complement to M-Stage. After all, using an ADL, application deployers are able to comprehend the overall aspect-component composition, facilitating a better understanding and easier verification of the application. This is of course an important software engineering quality that improves the safety and robustness when deploying aspects. M-Stage could also be nicely complemented by model-driven middleware (e.g. [25,26,24] In model-driven middleware, multiple design models of aspects and applications can be specified, composed and possibly verified. Once composed, these models can be automatically synthesized to deployment descriptors for a specific (non-AO) middleware platform of choice [25] or to middleware implementations itself [26]. An approach with similar goals to M-Stage in this context is the CoSMIC and the DAnCE toolsets [25]. These toolsets specifically address crosscutting deployment and configuration concerns of distributed real-time and embedded systems. The difference between M-Stage and CoSMIC/DAnCE is that the latter targets system life-cycle challenges in standard middleware, while M-Stage addresses reuse problems in AO middleware.

5 Conclusion

M-Stage is an aspect-component model that offers reuse and refinement of compositions in distributed applications that are built on AO middleware. Key elements in M-Stage are its support for multi-stage composition and its multi-stage join point model. Multi-stage composition supports modularization and refinement of AO composition logic across multiple composition and deployment stages. The multi-stage join point model enables abstraction of certain context information during the early composition stages. We have illustrated the power of M-Stage by applying the model in a realistic distributed application where we analyzed the reuse and adaptation potential of the M-Stage model.

References

1. Szyperski, C.: Component software: beyond object-oriented programming, 2nd edn. ACM Press/Addison-Wesley
2. Heineman, G., Councill, W.: Component-based Software Engineering. Addison-Wesley
3. Kiczales, G.: Aspect-Oriented Programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, Springer, Heidelberg (1997)
4. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.: An Overview of AspectJ. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, Springer, Heidelberg (2001)
5. Hilsdale, E., Hugunin, J.: Advice Weaving in AspectJ. In: Proc. AOSD 2004
6. Filman, R., Elrad, T., Clarke, S., Aksit, M.: Aspect-Oriented Software Development. Addison-Wesley, Reading (2004)

7. Pawlak, R., Seinturier, L., Duchien, L., Florin, G.: JAC: A Flexible Solution for Aspect-oriented Programming in Java. In: Yonezawa, A., Matsuoka, S. (eds.) Reflection 2001. LNCS, vol. 2192, Springer, Heidelberg (2001)
8. Fleury, M., Reverbel, F.: The JBoss extensible server. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, Springer, Heidelberg (2003)
9. Lagaisse, B., Joosen, W.: True and Transparent Distributed Composition of Aspect-Components. In: van Steen, M., Henning, M. (eds.) Middleware 2006. LNCS, vol. 4290, Springer, Heidelberg (2006)
10. Pinto, M., Fuentes, L., Troya, J.M.: A Dynamic Component and Aspect-Oriented Platform. The Computer Journal (2005)
11. Navarro, L.D.B., Südholt, M., Vanderperren, W., De Fraine, B., Suvée, D.: Explicitly distributed AOP using AWED. In: Proc. AOSD 2006 (2006)
12. Cohen, T., Gil, J.Y.: AspectJ2EE = AOP + J2EE: Towards an Aspect Based, Programmable and Extensible Middleware Framework. In: Odersky, M. (ed.) ECOOP 2004. LNCS, vol. 3086, Springer, Heidelberg (2004)
13. JBoss AOP homepage, http://labs.jboss.com/jbossaop
14. Spring website, http://www.springframework.org/
15. AspectWerkz homepage, http://aspectwerkz.codehaus.org/
16. Batory, D.S., Sarvela, J.N., Rauschmayer, A.: Scaling Step-Wise Refinement. In: Proc. ICSE 2003, pp. 187–197 (2003)
17. Nicoara, A., Alonso, G.: Dynamic AOP with PROSE. In: ASMEA 2005. Proc. of International Workshop on Adaptive and Self-Managing Enterprise Applications (June 2005)
18. Wohlstadter, E., Devanbu, P.T.: Aspect-Oriented Development of Crosscutting Features in Distributed, Heterogeneous Systems. In: Transactions of Aspect-Oriented Software Development II, pp. 69–10 (2006)
19. Wohlstadter, E., Tai, S., Mikalsen, T.A., Rouvellou, I., Devanbu, P.: GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions. In: ICSE 2004, pp. 189–199 (2004)
20. Nishizawa, M., Chiba, S., Tatsubori, M.: Remote pointcut: a language construct for distributed AOP. In: AOSD 2004, pp. 7–15 (2004)
21. Atkinson, C., Kühne, T.: Aspect-Oriented Development with Stratified Frameworks. IEEE Software 20(1), 81–89 (2003)
22. Wang, N., Schmidt, D.C., O'Ryan, C.: Overview of the CORBA Component Model. Component-based software engineering: putting the pieces together, pp.557–571 (2001)
23. Monson-Haefel, R.: Enterprise JavaBeans, 3rd edn. O'Reilly (September 2001)
24. Gray, J., Bapty, T., Neema, S., Schmidt, D.C., Gokhale, A., Natarajan, B.: An approach for supporting aspect-oriented domain modeling. In: Pfenning, F., Smaragdakis, Y. (eds.) GPCE 2003. LNCS, vol. 2830, pp. 151–168. Springer, Heidelberg (2003)
25. Deng, G., Schmidt, D.C., Gokhale, A.S.: Addressing crosscutting deployment and configuration concerns of distributed real-time and embedded systems via aspect-oriented and model-driven software development. In: Proc. ICSE 2006, pp. 811–814 (2006)
26. Zhang, C., Gao, D., Jacobsen, H.: Generic Middleware Substrate Through Modelware. In: Alonso, G. (ed.) Middleware 2005. LNCS, vol. 3790, pp. 314–333. Springer, Heidelberg (2005)
27. Pinto, M., Fuentes, L., Troya, J.M.: DAOP-ADL: An architecture description language for dynamic component and aspect-based development. In: Pfenning, F., Smaragdakis, Y. (eds.) GPCE 2003. LNCS, vol. 2830, pp. 118–137. Springer, Heidelberg (2003)
28. Pessemier, N., Seinturier, L., Coupaye, T., Duchien, L.: A Model for Developing Component-based and Aspect-oriented Systems. In: Löwe, W., Südholt, M. (eds.) SC 2006. LNCS, vol. 4089, Springer, Heidelberg (2006)
29. Shaw, M., Garlan, D.: Software Architecture: Perspective on an Emerging Discipline. Prentice-Hall, Englewood Cliffs (1996)

An Eclipse-Based Tool for Symbolic Debugging of Distributed Object Systems[⋆]

Giuliano Mega and Fabio Kon

Department of Computer Science, University of São Paulo, Brazil
{giuliano, kon}@ime.usp.br
http://god.incubadora.fapesp.br

Abstract. After over thirty years of distributed computing, debugging distributed applications is still regarded as a difficult task. While it could be argued that this condition stems from the complexity of distributed executions, the fast pace of evolution witnessed with distributed computing technologies has also played its role by shortening the life-span of many useful debugging tools. In this paper we present an extensible Eclipse-based tool which brings distributed threads and symbolic debuggers together, resulting in a simple and useful debugging aid. This extensible tool is based on a technique that is supported by elements that are common to synchronous-call middleware implementations, making it a suitable candidate for surviving technology evolution.

1 Introduction

After over thirty years of distributed computing, debugging distributed applications is still a difficult task. While it is true that this could be partially blamed on the fact that distributed executions are complex and difficult to handle, a major contributing factor to this situation has been the fast pace at which new distributed computing technologies (including hardware, middleware, and operating systems) have emerged, making the life-span of debugging tools somewhat short. Be as it may, the net result is the noticeable lack of a set of effective, widely adopted debugging tools, even on mainstream middleware platforms.

We are not the first ones to identify heterogeneity as a major contributor to the slow progress witnessed with mainstream debugging tools. Cheng had already identified it in 1994 [2], and so had Pancake [15], as well as many other researchers and industry specialists. This points out to the fact that portability - not just among hardware platforms, but also among middleware and operating systems - is of paramount importance if a tool is to be useful within today's highly heterogeneous environments, and also if this tool is to remain useful for more than a couple of seasons. One way to achieve portability is through standardization. In the High-Performance computing arena, there have been some rather important efforts - like the High Performance Debugging (HPDF) forum,

[⋆] This work was partially supported by a grant from CAPES-Brazil, by an Eclipse Innovation Grant, and by a Google Summer of Code grant.

the OMIS project, and the Parallel Tools Consortium - which attempted to push the development of a set of standards for debugging and performance analysis tools. These efforts were mainly targeted at the needs of the high performance computing community, however, and, to the best of our knowledge, no such efforts have ever been attempted within the distributed object community. This puts distributed object application developers in a difficult situation as far as compatibility, longevity and, by consequence, availability and usability of debugging tools is concerned.

In this paper, we present a new framework for distributed debugging that can be applied to multiple middleware platforms and programming languages. The framework has been developed as an Eclipse plugin, which allowed us to reuse the rich debugger user interface provided by the platform. This framework is validated through a Java/CORBA instantiation that supports features such as distributed application launching, breakpointing, distributed stack and state inspection, multiple extended stepping modes, and distributed deadlock and stack overflow detection, among other features.

Motivation: Distributed Object Middleware and Symbolic Debugging

From a historical perspective, one of the most popular abstractions for inter-process communication in distributed systems has been that of the remote procedure call (RPC) [13]. RPCs have been widely employed for over two decades, either in a procedure-oriented fashion or, somewhat more recently, as remote method invocations in object-oriented middleware systems such as CORBA, DCOM, Java/RMI and .NET/Remoting.

In an equal ground as far as popularity is concerned, we have the symbolic or source-level debuggers (like GDB [17], for instance). Symbolic debuggers have been around for almost as long as higher-level languages themselves, and still constitute one of the most widely used and accepted tools for helping programmers remove bugs from programs. We believe that this popularity can be explained by the fact that symbolic debuggers are the only kind of debugging tool, apart from the print statement, that is available for almost every language, runtime environment, operating system, and hardware currently in existence. In a sense, we could say there is a standard at work here - albeit not a formal one. Distributed applications (including Distributed Object Applications) are, in fact, frequently debugged with symbolic debuggers capable of operating remotely.

Our approach to the distributed debugging problem attempts to unite the essence of what makes synchronous calls and symbolic debuggers so convenient. At the same time, we try to augment the latter with functionality so that they can become more useful in the context of multithreaded, distributed object applications (DOAs), while keeping in mind the goals of extensibility, portability and simplicity. The philosophy that actually backs this whole work is already quite simple - we want to bring symbolic debugging of distributed object applications as close as possible to the debugging of centralized, multithreaded applications. We also want to support the user of Distributed Object Computing (DOC) middleware in accomplishing debugging tasks that result from the insidious complexities of synchronous calls; that is, we wish to help the user of DOC middleware to overcome some of its inherent complexities. That said, symbolic debuggers are convenient for the following reasons:

1. Ubiquity: it is very rare to see a language development toolkit shipping without a symbolic debugger – this is even more true with mainstream languages. It is therefore not unreasonable to take for granted that the languages and runtime environments on top of which a heterogeneous distributed object system will be built on will have symbolic debuggers available for them.
2. Cognitive appeal: the visualization mechanism used by symbolic debuggers - animation over the source code - is quite intuitive, and matches the (low-level) mental images that the programmer produces while coding. Although this visualization mechanism does not scale particularly well, it is something developers are familiarized with, and it is definitely helpful. Synchronous-call DOC middleware present the distributed system as a collection of virtual threads that are capable of spanning multiple machines. We will call these *distributed threads* (or DTs). Symbolic debuggers and the underlying execution environment, including its libraries, however, are myopic to such high-level constructs. This brings on a number of issues (some of which are discussed in [10]), which we present in the following paragraphs:

1. Abstraction mismatch: middleware platforms allow developers to treat remote and local objects similarly. Middleware implementations accomplish this transparency by replacing the implementation of remote object references accessible from clients with code that is generated automatically, either during compilation or at runtime. The problem happens when such an application is subjected to the eye of a symbolic debugger, as all of this automatically generated code will be exposed to the user, together with the code of the middleware platform, defeating the benefits of transparency. Not only that, but the user will also not be able to track the flow of control of his application (by stepping) into remote objects like he does with local objects, simply because symbolic debuggers are not aware of this arrangement. In other words, the view of the underlying execution environment that is provided by the debugger does not match the abstract view provided by the DOC middleware.
2. Causal relationships are not properly captured: capturing causality [16] is a task that is out of scope of most symbolic debuggers. For DOC middleware, this means that users will not be able to see the order in which events have happened. Also, they will not be able to see which local threads participate in which distributed threads.
3. Distributed and self-deadlocks: Like with multithreaded applications, distributed threads can deadlock when acquiring the same locks in different orders. Distributed deadlocks can be tough to spot with plain symbolic debuggers. Also, although a distributed thread logically represent a single thread, it is actually composed of multiple, “local” threads. Since the concurrency mechanisms of the underlying execution environment are normally oblivious to the existence

of distributed threads, this may lead to situations where a distributed thread deadlocks with itself (we call that a *self-deadlock*).

Besides the issues already mentioned, there are a number of other issues that constitute classical problems in distributed debugging [7]. For the sake of completeness, we outline these as follows:

1. Non-determinism and the probe effect: distributed executions are intrinsically non-deterministic due to their partially ordered nature [12]. The interleaving of instrumentation code with application code for gathering of run-time information may lead to timing distortions which affect (and bias) the partially ordered, distributed execution. This is known as the probe effect [5] and may lead to "heisenbugs" (erroneous behavior that disappear under observation).
2. Maze effect: the maze effect [4] manifests itself whenever the user of a debugging tool is overwhelmed with information. Representations of the distributed execution that are too fine grained are one common cause. Tools that are unable to selectively display execution information using some relevance criteria, thus overwhelming the user with data, are another cause. We believe synchronous calls and symbolic debuggers to be a good starting point for three main reasons: their popularity among developers, their pervasiveness among middleware platforms/programming languages, and because the resulting tool – a distributed symbolic debugger – is something most developers will be familiar with, even though they might have never seen or used one. The source code for all implementation efforts described in this article can be obtained as free software at http://god.incubadora.fapesp.br.

2 Debugging with Distributed Threads

This section begins by attempting to describe in more precise terms *what* are distributed threads (DTs), and by laying a formal framework that tells how we expect them to behave. We then proceed by describing how DTs can help us cope with a number of debugging issues, and present a general idea of how a tool that explores them works.

Before that, however, we would like to reach an informal agreement on what a non-distributed – i.e., a "local" – thread is. A local thread is a "regular" thread. Local threads are restricted to a single addressing space and to a single processing node. Examples of local threads include traditional lightweight processes, such as those implemented at the kernel level in operating systems like GNU/Linux and Microsoft Windows, for instance, as well as heavyweight processes and "green" (non-OS, usually non-preemptive) thread implementations. This informal definition should be enough for our purposes.

For the sake of simplicity, we assume time to be a continuous and linear set of instants that is isomorphic to the set of real numbers $\mathbb{R}$ (i.e., we represent time instants as real numbers). For every distributed computation C, we assume that there are two real-valued instants t_C^s, t_C^d that represent the instants in which C begins and ends, respectively. We say that $[t_C^s, t_C^d]$ is the *lifetime interval* of C. We call L_C the set of all local threads that ever took part in C. Since C has a lifetime interval, all local threads in L_C also have lifetime intervals. Therefore, we can assign a pair of real-valued time instants t_l^s and t_l^d to each local thread $l \in L_C$, such that these values represent the instants when local thread l starts and dies, respectively, and $[t_l^s, t_l^d] \subseteq [t_C^s, t_C^d]$.

Definition 1 (Distributed Thread Snapshot). *A distributed thread snapshot over a distributed computation C is defined to be a sequence $s = \{l_1, ..., l_m\}$ of local threads, where $l_i \in L_C$ ($i \in \mathbb{N}$ and $1 \leq i \leq m$).*

By analogy to L_C, we will define S_C to be the set of all snapshots that can be formed with threads drawn from L_C. We are now ready to define a distributed thread.

Definition 2 (Distributed Thread). *Let C be a distributed computation. A distributed thread T is a sequence of snapshots $\{s_{t_1}, ..., s_{t_n}\}$, where each t_i ($1 \leq i \leq n, i \in \mathbb{N}$) represents a real-valued time instant ($t_i \in \mathbb{R}$), and all s_{t_i} are drawn from S_C. Also, if $i < j$, then $t_i < t_j$; that is, T is totally ordered with respect to time. For every distributed thread $T = \{s_{t_1}, ..., s_{t_n}\}$, the following properties must hold:*

1. *There exists a local thread $l_1 \in L_C$ such that $s_{t_1} = \{l_1\}$, and l_1 is the first element of s_{t_i}, for all i. We will say that l_1 is the* **base** *of distributed thread T.*
2. *Let $i \in \mathbb{N}$ and $1 < i \leq n$, and let $s_{t_{i-1}} = \{l_1, ..., l_m\}$. Then, in the absence of failure, exactly one of the following must hold:*
   (a) $s_{t_i} = \{l_1, ..., l_m, l_{m+1}\}$, where $l_{m+1} \in L_C$. In this case, l_m have initiated a remote request at instant $t_{i-1} + \delta \leq t_i$ (where $\delta \in \mathbb{R}$ and $\delta > 0$), and thread l_{m+1} have begun handling this remote request at instant t_i.
   (b) $s_{t_i} = \{l_1, ..., l_{m-1}\}$. In this case, thread l_m has finished handling, at instant t_i, the remote request that had been previously initiated by l_{m-1}.
3. *Let t_T^s and t_T^d be the instants in which T starts and dies, respectively. Then $t_1 = t_{l_1}^s = t_T^s$ and $t_n < t_{l_1}^d = t_T^d$.*

Let $s_{t_i} = \{l_1, ..., l_m\} \in T$. We say l_m is the *head of T* at instant t_i. We also say that the threads $l_1, ..., l_m$ *participate in T* at instant t_i. Definition 2 has a number of interesting implications. First, for every local thread $l \in L_C$, we have a distributed thread T such that **(1)** T starts and dies with l, and **(2)** l is the base of T. Second, DT snapshots can be interpreted as follows. Let $s_{t_i} = \{l_1, ..., l_n\}$, **(1)** if $1 < i \leq n$, then l_i is handling a request that has been initiated by l_{i-1}, and **(2)** if $i < n$, then l_i is blocked in a remote call that is being handled by l_{i+1}.

We shall call a snapshot that contains a single local thread a *trivial snapshot*. All distributed threads begin with a trivial snapshot, and may contain several trivial snapshots (all identical) along its snapshot set. Fig. 1(a) shows a DT and part of its snapshot set (relevant state shifts) as it progresses through a three-node call chain. Lastly, our notion of a valid set of DTs shall be bound by a rule we call "the single participation rule". This rule, as we will see, shall constrain the class of middleware implementations that our technique applies to. In order to make the definition less complicated, we will define two auxiliary concepts,

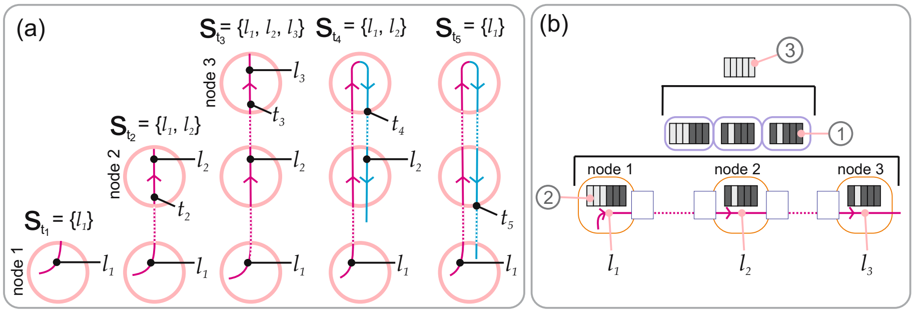


Fig. 1. (a) DT and its sequence of snapshots during part of a three-node call chain. (b) Assembly of the virtual stack from a snapshot.

expressed in Def. 3 and Def. 4. What Def. 3 says is that the state of a distributed thread T, at an arbitrary instant x, is either empty (if x falls off the lifetime interval $[t^s_T, t^d_T]$ of T) or it corresponds to the last snapshot of T up to instant x.

Definition 3 (State of a Distributed Thread). *Let D_C be the set of all distributed threads that ever existed in C and let $T = \{s_{t_1}, ..., s_{t_n}\} \in D_C$ be one of these distributed threads. Also, let S_C be defined as before. The state of a distributed thread T at instant $x \in \mathbb{R}$ is given by the function $f : D_C \times \mathbb{R} \rightarrow S_C$ where:*

$$f(T, x) = \begin{cases} s_{t_i}, & \textit{if } t_i \leq x < t_{i+1} \textit{ and } 1 \leq i < n, \textit{ or} \\ & t_i \leq x < t^d_T \textit{ and } i = n \\ \emptyset, & \textit{otherwise} \end{cases}$$

Definition 4. *Let s_1 and s_2 be two distributed thread snapshots. We say that s_1 is a subsnapshot of s_2, or that $s_1 \rightarrow s_2$, if and only if s_1 is a suffix of s_2.*

The “single participation rule” attempts to establish the situations in which it is valid for a local thread l to participate in the state of more than one distributed thread, simultaneously. Mainly, we would like to express that a local thread l cannot simultaneously participate in the state of more than one distributed thread, except under some very special circumstances. These circumstances will be characterized with the help of the following remark:

Let $s_{t_i} = \{l_1, l_2, ..., l_n\}$ be a nontrivial snapshot of a distributed thread T. Then s_{t_i} can be expressed as $\{l_1\} \cup f(T', t_i)$, where T' is the distributed thread whose base is l_2. That is $f(T', t_i) \rightarrow f(T, t_i)$.

To give a more concrete example of the meaning of this remark, let T be a distributed thread, and suppose $f(T, t) = \{l_1, l_2, l_3\}$ at some instant $t \in \mathbb{R}$. The remark shows that, if we take a local thread $l_2 \in f(T, t)$, then this local thread participates – simultaneously – in the states of all distributed threads that can be formed by removing prefixes of size less than 2 from $f(T, t)$. In our example, l_2 will participate in $f(T, t) = \{l_1, l_2, l_3\}$ and in $f(T', t) = \{l_2, l_3\}$, where T' is the distributed thread whose base is l_2 – hence l_2 participates in the state of more than one distributed thread simultaneously. Those will be the only situations where it will be allowable for a local thread to participate in the state of more than one distributed thread simultaneously. Therefore:

"Single Participation Rule": Let $\{s_1, ..., s_n\}$ be the set of snapshots in which a local thread l participates at an instant t. The single participation rule is obeyed if we can find a permutation π of $\{1, ..., n\}$ such that $s_{\pi(1)} \rightarrow ... \rightarrow s_{\pi(n)}$.

The work described in this paper applies, in general, to distributed computations whose set of DTs conform to the single participation rule. Though most middleware implementations do produce compliant executions, some real-time ORBs [8] may not – but then again, symbolic debuggers are usually regarded as being far too intrusive for real-time systems. DTs play an important role in the achievement of our goal, because they make the necessary link between RMI-based distributed object systems and symbolic debuggers. The whole idea behind a symbolic debugger that leverages DTs is that it may present the execution as a collection of states on DTs, instead of on loosely-coupled local threads.

A more practical view of a distributed thread – as it is displayed by our debugger – is shown in Fig. 1(b). The depicted snapshot **(1)** is composed of three local threads (l_1, l_2, and l_3), which are represented along with their call stacks **(2)**. The darker stack frames represent calls into middleware code, whilst the lighter frames represent calls into user code. The debugger strips out the darker frames, assembling and presenting to the user a virtual stack **(3)**, which is composed of user code only. The debugger then allows the user to interact with this "virtual" thread as he does with regular threads – stepping, resuming, suspending, inspecting, etc. – in debuggers such as GDB [17].

3 A Distributed-Thread-Based Debugger

Now that we have presented our motivation and characterized in precise terms how the distributed threads we intend to deal with should behave, we will present the actual tool we have built, that leverages them for debugging.

3.1 Representing and Tracking Distributed Threads

The first step to presenting the distributed system as a collection of distributed threads is being able to track them. There are a multitude of possible approaches to the problem, but those mostly vary between how much of the distributed thread representation will stay at the debugger side (meta-level) and how much will stay at the debuggee side (base-level). Our approach begins by constructing a representation of the distributed threads that is accessible at the base-level.

This representation is inspired on the way distributed transactions are typically identified, and also by the work of Li [9]. In our approach, each local thread that participates in a distributed computation is uniquely identified by a two-part, 48 bit id. The first 16-bits uniquely identify the node to which the local thread belongs to. The remaining 32-bits are drawn from a local sequential counter. 48 bit ids allow us to identify enough nodes/local threads while

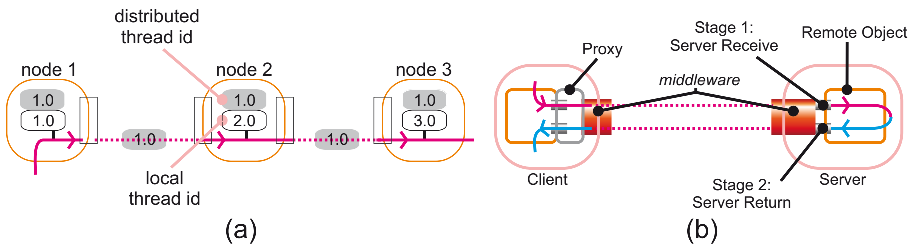


Fig. 2. (a) Propagation of the id of a local thread, tainting subsequent threads and characterizing a distributed thread. (b) Two-stage tracking protocol.

keeping overhead small, but the actual id length can be tuned. Whenever a local thread that is part of a single trivial snapshot initiates a chain of remote calls, its id is propagated, "tainting" all subsequent threads in this call chain, as shown in Fig. 2(a). As for the actual tracking of distributed threads, we just have to remember that for each snapshot, there is always a single head. The remaining threads are all blocked, and, if we assume a failure-free scenario (we will lift this restriction in a moment) we can expect that these blocked threads will not produce any "interesting" changes until they become heads again – that is, until the current head finishes handling the remote request that has been initiated by the local thread that immediately precedes it in the current snapshot. More precisely speaking, local thread "starts handling a remote request" when the middleware calls into the remote object code for the first time during the handling of this remote request, causing a frame, which we will call the *entry frame*, to be pushed into the call stack of the server-side, local thread. Based on this information, it is not hard to come up with a simple tracking protocol that can be used to reconstruct the trajectory of a distributed thread. Our protocol, shown in Fig. 2(b), is composed of two stages: **(1) Server Receive** and **(2) Server Return**, which are triggered when an *entry frame* is pushed, and popped, respectively. Both stages capture the id of the distributed thread, and the id of the local thread.

3.2 Interactivity and Synchronicity

Our intention is to extend a symbolic debugger so that it may handle distributed threads as naturally as it handles local threads. Symbolic debuggers are on-line [7], interactive debuggers by nature, and their ability to operate (e.g., view, suspend, step, resume, inspect) on threads is amongst the most fundamental ones. The problem with interactive operations is that they operate on "live" entities. Debuggers, however, act as observers of computations – they will merely reconstruct an approximation of the state of the application based on information that is sent from instrumentation probes (*local agents* in Fig. 3(a)) that are deployed with the application. Therefore, there is always the chance that the state of the execution as observed by the debugger will not correspond to the actual state of the application, either due to network latency, or because the state

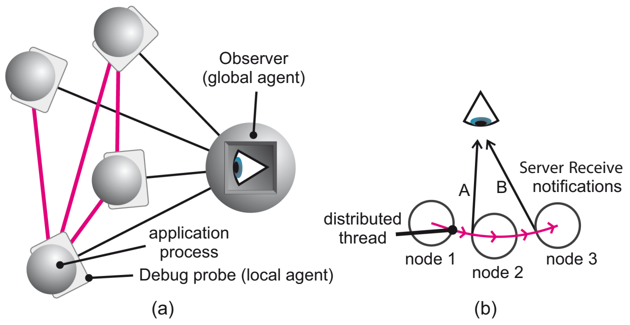


Fig. 3. (a) High-level architectural view of our distributed debugger. (b) Causally related events in a single call chain.

change cannot be immediately detected/reported (as when the entire machine that contains the debuggee crashes). This kind of situation is, for obvious reasons, much more common in scenarios where debugger and debuggee are separated by a network. From our experience, debugger implementations may handle these situations in one of two ways: **(1)** allow state drift to go unbounded, knowing that interactive operations might fail because the "live" entity at which the operation is directed may no longer exist, or may transition to a state where the operation is no longer allowed; or, **(2)** bound the drift by synchronizing debugger and debuggee at certain key state transitions, therefore eliminating operation failures that arise from the sort of race condition described in **(1)**.

Alternative **(2)** is accomplished by having the debug probes halt the execution of the local processes at some key state transitions to make sure that the debugger (global agent in Fig. 3(a)) has registered these transitions before proceeding. This does not help, however, in cases where the operation fails because the remote node died unexpectedly – in fact, node death due to crashes is one of the only kinds of state changes that cannot, ever, be reported in a synchronous fashion. Basic guidelines for scalable distributed systems dictate that we should take the first option whenever possible, reducing the amount of synchronous reports to a minimum. The particular issue we faced was how to support user controlled *suspend* operations on distributed threads reliably. A precondition to being able to suspend a distributed thread at an arbitrary instant is knowing the exact position of its head, also at an arbitrary instant – therefore this relates directly to *how* we may consume the information produced by the protocol described in Sec. 3.1. The problem lies in the fact that, if we track the head asynchronously, then we have no way of knowing if the knowledge of the global agent with respect to the current head of a given distributed thread is stale or not. This could lead to a situation where the debugger is constantly behind the actual state for a given distributed thread, making support for suspend inefficient and unreliable. Therefore, in order to provide adequate support for suspend, we opted for synchronously tracking the head; that is, the reporting of *Server Receive* and *Server Return* events will cause the ongoing request to halt until the global agent is known to have updated its knowledge about the new head.

One useful consequence of capturing these state changes in a synchronous fashion is that if two events are causally related (like events A and B in Fig. 3(b)) then the global agent will never observe an inconsistent ordering, simply because B can not happen while A is not allowed to complete. Therefore, we can get away without timestamping, and still trust that the results will be correct.

3.3 Node and Link Failures

So far we have been discussing how to track distributed threads in the absence of failures. When failures are introduced (link failures and node crashes) it is possible that threads which were known to be in the middle of a snapshot begin unblocking and producing snapshot-changing events, thus violating the expected behavior (as by Def. 2) for distributed threads. To deal with such situations, we will introduce an extension to Def. 2 which allows distributed threads to be "split" under certain circumstances. Before that, however, we must define what we consider to be a "normal" unblocking for a local thread.

Definition 5 (Normal Unblocking of a Local Thread). *A local thread l_i that is blocked in a remote call is said to have unblocked normally if its unblocking results from the processing of a reply message, that has been sent by the server which contained the thread that handled the request initiated by l_i, informing that this remote request has completed (either successfully or unsuccessfully).*

Therefore, a blocked local thread unblocks non-normally whenever its unblocking can not be causally traced to a reply message from the server. Now let $f(T, t) = \{l_1, ..., l_n\}$ be the state of a distributed thread T at instant t. Also, let T' be the distributed thread whose base is l_{i+1}, where $l_{i+1} \in f(T, t)$, and let $\epsilon \in \mathbb{R}, \epsilon > 0$. We shall establish that:

– If thread l_i unblocks non-normally at instant $t + \epsilon$, then $f(T, t + \epsilon) = \{l_1, ..., l_i\}$, and $f(T', t + \epsilon) = \{l_{i+1}, ..., l_n\}$.
– If thread l_i is known to be dead at instant $t+\epsilon$, then $f(T, t+\epsilon) = \{l_1, ..., l_{i-1}\}$ and $f(T', t + \epsilon) = \{l_{i+1}, ..., l_n\}$.

In both cases, we say that T has been "split". Thread splits are enough for us to reorganize information whenever a node or link fails for whatever reason, leaving broken distributed threads behind. The user will be notified whenever a split occurs (as these are always errors), and the debugger will do its best to relate the split to its cause (mainly by checking whether the node that is adjacent to the split is still alive).

3.4 Debugger Architecture and Implementation

So far we have discussed our debugger at a fairly high and conceptual level. In this section we discuss its architecture, and some key aspects of its implementation. A general layout of the architecture has already been given in Fig. 3(a). The debugger can be roughly decomposed into two types of participants – the

global agent, which is the module responsible for assembling execution information, presenting them to the user, and accepting interactive commands, and the **local agents**, or debug probes, which are responsible for collecting runtime information and interacting with the distributed system processes on behalf of the global agent (or any other client). This centralized architecture is a natural result of the fact that at some point there must be one single observer, who has a global view of the distributed computation.

The Local Agents are composed of a combination of standard symbolic debuggers and custom instrumentation code that is injected by our extensions. This custom instrumentation code implements the thread id propagation scheme described in Sec. 3.1, the tracking protocol, and other assorted functionalities required by the debugger. As shown in Fig. 4(a), local agents use two distinct wire protocols – one that is specific to the symbolic debugger in use (which will be used for setting breakpoints, controlling local threads, getting local state information, etc.), and another one, which is language-independent, that will be used to convey the information required by the thread tracking protocol of Sec. 3.1 (we call this protocol DDWP, or Distributed Debugging Wire Protocol). A simplified schematic view of the actual tracking mechanism, in its Java/CORBA version, is presented in Fig. 4(b). The Java tracking mechanism of Fig. 4 **(b)** is

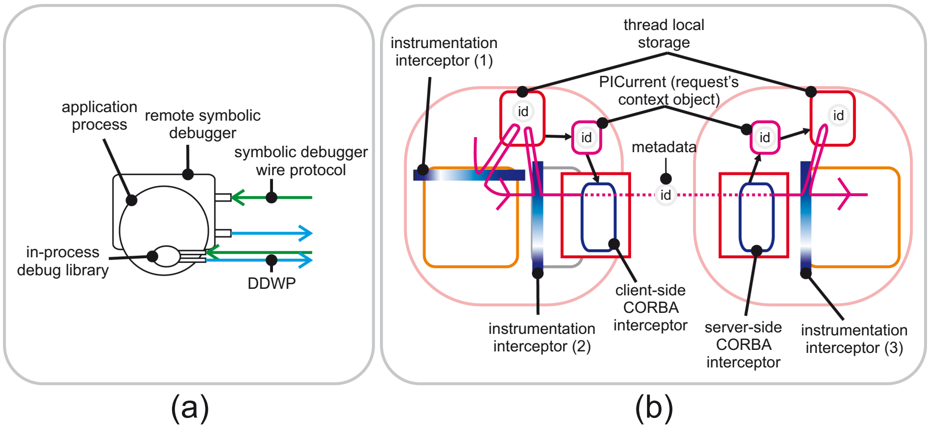

application process
remote symbolic debugger
symbolic debugger wire protocol
in-process debug library
DDWP
(a)
instrumentation interceptor (1)
thread local storage
PICurrent (request's context object)
id
metadata
client-side CORBA interceptor
instrumentation interceptor (2)
server-side CORBA interceptor
instrumentation interceptor (3)
(b)

Fig. 4. (a) Anatomy of the local agent. (b) Tracking mechanism for Java/CORBA.

implemented through a combination of thread-local storage, CORBA portable interceptors, and custom instrumentation interceptors, which are inserted at runtime with the help of a Java transformation agent, and the Bytecode Engineering Library [1]. The first custom interceptor is inserted at the beginning of each *Runnable#run()* method, and is responsible for assigning the unique two-part id described in Sec. 3.1 to each local thread as soon as it is started, as well as for enrolling the local threads in a registration protocol which is required for the mapping of numeric ids to *ThreadReference* mirrors provided by the Java Debug Interface (JDI) [18]. The second (inserted at each CORBA stub) and third (inserted at each CORBA servant) instrumentation interceptors will bridge the thread-local storage and the CORBA portable interceptors, allowing us to pass on the required ids which each request.

Apart from its use in the tracking mechanism, the CORBA and Instrumentation interceptors are also used for the detection of non-mediated recursive calls, and abnormal unblockings of blocked local threads (Sec. 3.3). For the detection of recursive calls, we simply insert a token in the *PICurrent* object whenever a request passes through a CORBA interceptor. The instrumentation interceptors inserted at the remote objects **(3)** will always test for the presence of this token. If it is found, the interceptor will remove it, generate a *Server Receive* event, and initialize a thread-local counter to zero. Subsequent calls to remote object implementations that are not mediated by the ORB will trigger the instrumentation interceptors which, failing to see the token, will just increment the thread-local counter, without generating further *Server Receive* events. Whenever one of these instrumented methods return (either due to a normal return, or due to an exception), the instrumentation interceptor **(3)** will be activated again, and the thread-local counter will be decremented. When the counter reaches zero, the interceptor will know that the current stack frame is an entry frame, and will generate a *Server Return* event to signal that the current head has changed.

The mechanism for testing for abnormal unblockings is also token-based. Recall from Def. 5 that unblockings occur whenever a given thread l_i unblocks due to a reason other than the client-side middleware getting a regular reply message. The two most common causes behind abnormal unblockings are link and node failures, which are not distinguishable from the point of view of the client. We detect abnormal unblockings by sending a single-bit token with each reply. This single-bit token is inserted into the request service context by the server-side CORBA interceptor shortly before the reply is sent, and loaded into the *PICurrent* object by the client-side CORBA interceptor when the reply is received. If there is no reply message, however, the token will never get to the instrumentation interceptor **(1)**, indicating that an abnormal unblocking has occurred. Whenever that happens, the instrumentation interceptor **(1)** will (synchronously) notify the global agent, which will perform the appropriate distributed thread split and notify the user of the erroneous behavior.

The Global Agent: has many responsibilities. It has to control and manage the distributed processes, it has to combine the partial state information provided by each of the local agents, and it has to handle user interaction. According to the notions on computational reflection laid down by Maes [11], symbolic debuggers can be seen as meta-systems whose domain are the execution environments of the debugged processes. Among other things, this means that debuggers should have access to structures that represent (aspects of) the execution environments of such debugged processes. In order to keep things simple, we decided to take the JDI [18] approach and reify distributed threads at the symbolic debugger level. There were, however, many other issues we needed to resolve.

The elements that compose the execution environment of a distributed application may come from many environments. Reifying the environment of the distributed object system means coming up with an object model capable of accommodating this heterogeneity. Also, since we are worried with extensibility, we would like to have a model that is capable of accommodating new environments with relative ease. Coming up with such a model from the ground up, however, requires time and experience. Fortunately there is already one mature, open source and widely developed debug model which has proven to accommodate heterogeneity, and which would be a perfect fit for our own debugger: the Eclipse debug model [19]. Eclipse is a well-known, extensible environment for building Rich Client Applications. Its debug model has successfully accommodated reified versions of the main elements of Java, C++, Python, Ruby and many other execution environments. Based on that observation, we decided to implement our distributed-thread-based debugger as a set of extensions to the Eclipse debug framework. Our extensions are depicted as the grey areas in Fig. 5. Our main contribution to the Eclipse debug framework has been a

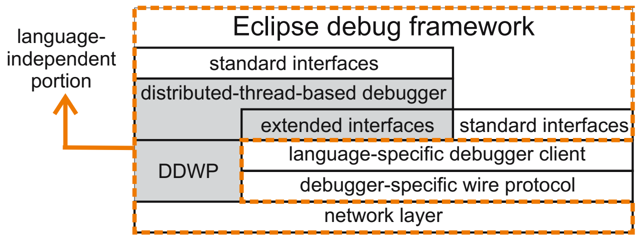

language-independent portion
Eclipse debug framework
standard interfaces
distributed-thread-based debugger
extended interfaces
standard interfaces
DDWP
language-specific debugger client
debugger-specific wire protocol
network layer

Fig. 5. Layered architecture of the global agent

language-independent, distributed thread debugger, and set of extensions to the regular Eclipse *IThread* interface. The contribution of the Eclipse platform to our project, however, has been also very rich – a collection of ready-to-use debugging solutions that could be realistically adapted to work with our debugger. Adapting an existing Eclipse debugging client amounts to implementing our extended interfaces.

Applicability – middleware, language, and debugger requirements: Now that we have discussed some of the key aspects of our implementation, we are in position to make an assessment on some of the requirements imposed by the technique. This should point us toward some answers to the question that matter the most: how difficult it is to actually port our debugging machinery to other languages/runtime environments, as well as other middleware systems:

1. Application must be based on distributed objects. To take advantage of the distributed thread abstraction, it should also use synchronous calls. Asynchronous calls are supported, but benefits are less clear.
2. The target middleware should allow context information (metadata) to be passed with each request. This is the only requirement imposed on the middleware apart from the use of distributed objects.

3. There must be a “standard” symbolic debugger available for the language, and we should be able to operate it remotely.
4. Object proxies (stubs) and remote objects (servants) must be instrumentable.
5. There must be a way to assign identifiers to each of the local threads that will take part in distributed threads.
6. For distributed deadlock detection, there must be a way for the global agent to know which locks are being held by which local threads.

Requirement 1 is actually a restatement of the type of systems that are the target of this work. Requirement 2 is fulfilled by almost every middleware implementation in use today. Also, it is not a hard requirement – we could get away with 2 by modifying the stub/skeleton generator to include an extra parameter, as in [9]. This would be, however, much more cumbersome. Requirement 3 is also rather reasonable, at least with mainstream languages. A “standard” symbolic debugger is, roughly speaking, a debugger that supports at least line breakpoints, step into, step over, step return, and source mapping capabilities – a feature set that is common to all symbolic debuggers we know of. Also, there must be a way to operate the symbolic debugger remotely, as with GDB [17], or the JPDA [18] debugger, for example. Requirements 4 through 6 are highly language-dependent. In Java – which could hardly be described as a language with strong reflective capabilities – we were able to implement the instrumentation mechanisms rather easily, thanks to Apache BCEL [1] and the Java instrumentation agents. In languages with more sophisticated reflective capabilities such as Python, Ruby, or Smalltalk, this should be even more straightforward. In a language like C++, the task would be made easier with software like OpenC++ [3].

3.5 More on Scalability and Correctness

There are two main sources of concern when the word “scalability” is applied to a distributed debugger: performance scalability (how many nodes can be debugged simultaneously before performance becomes an issue?), and visualization (how many nodes can be debugged simultaneously, before the maze effect takes over?). Our information visualization and navigation mechanism – based on the distributed thread abstraction – scales better than plain local threads. It allows the user to selectively focus his/her attention into what matters the most – the flow of control of his own application – while complementing this stripped down information with other kinds of error information, like detection of distributed deadlocks, node, and link failure. Therefore, from this point of view, our debugger has been built to scale better than conventional symbolic debuggers.

Performance scalability, on the other hand, has been one of our main sources of preoccupation from the start, due to our centralized architecture. This turned out to be less of an issue than initially thought, however, due to the dynamics of the updates produced by the local agents. The global agent keeps an internal table, where each entry contains state information for a distinct distributed thread. As we mentioned before, notifications are always synchronous. This means that, unless there is a thread split going on, updates to a single distributed thread

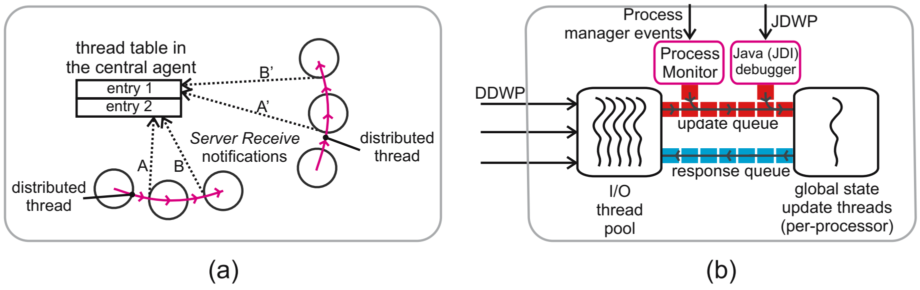


Fig. 6. (a) Dynamics of "regular" updates. (b) Processing in the global agent.

(table entry) will be performed one at a time. A hypothetical update scenario is shown in Fig. 6(a). Since these entries are disjoint, the updates can be performed concurrently, as long as we keep one updater thread per processor/core (Fig. 6(b)). Also, since updates are performed one at a time, contention on a single entry is non-existent in the absence of failure. Fig. 6(b) shows that the DDWP server is not the only source of state update events – the Java debugger (and other language debuggers), and the process monitor may also contribute with information. This information is related to thread and process lifecycle, and might be used by the updater threads to anticipate the occurrence of thread splits. Although we have not performed any conclusive tests on server scalability, we expect that its capacity to handle more nodes will increase as more processors are added, due to its simple design and due to the small amount of thread-shared state it contains. So far, the DDWP server has been capable of handling more than two dozen nodes without any noticeable performance degradation.

3.6 One-Click Launch, and Debugger for Testing

Most symbolic debuggers are capable of either instantiating or attaching to running processes with almost no burden on the user. In fact, with "modern" GUI-based debuggers, instantiating or attaching to running processes (either remote or local) can be, in most cases, a simple one-click operation (after previous configuration, of course). This is yet another point where centralized and distributed systems differ fundamentally: while centralized processes can be in one of two states – running or not running – distributed systems can be in many, partially running states, not all of which may work equally well. Take as example the instantiation of a simple client-server application, where the client makes a single request to the server. If the client is started before the server, then it is very likely that it will attempt to perform its request before the server has had the opportunity to properly initialize, resulting in failure. Simply ensuring that the server is started first, however, is not enough – the startup time for the client is probably much smaller than for the server, and the end results will be similar.

Keeping in line with our philosophy of making distributed debugging easier, we have developed a one-click instantiation facility that works with distributed systems as well. Taking our example and generalizing it a little, ensuring successful launching equals ensuring that certain processes are not launched until we are sure that all other processes it depends on are in a certain state (ready to take incoming communication). Defining proper state without getting application-specific could be a difficult matter. Fortunately, however, we have some powerful instrumentation and state inspection machinery at our disposal – the symbolic debugger itself. With that in mind, we developed a simple launch constraint language, which alleviates this issue by leveraging the knowledge obtainable by the symbolic debuggers. Mainly, this language allows the user to enter declarative statements like:

```
when <Name Srv> reaches org.jacorb.ORB:1278 launch <Srv1>,<Srv2>
when <SomeServer> reaches module1.Type2.line=‘‘ready_to_go’’ &
(module1.Type1.state=1 | module1.Type1.state=2) launch <Client>
```

These dependencies are mapped at runtime into edges in a DAG, which has all of the distributed system processes represented as vertices. The single-click run operation causes all processes with fan-in zero to be started. The remaining processes are launched as local predicates are satisfied. Something we quickly noticed is that this mechanism is very useful for writing automated distributed tests. It is a small part of the puzzle, of course, as it ensures only a deterministic launch sequence. That did not prevent it from being very useful, however, as we were writing distributed, automated integration tests for the debugger itself.

4 Related Work

There is a very vast body of literature on the subject of debugging (and testing) of concurrent programs, but most of this research has been directed at parallel systems. We have therefore selected three works which we consider to be most representative as far as the topics of debugging of distributed object applications and portable debugging are concerned.

OCI's OVATION: The Object Viewing & Analysis Tool for Integrated Object Networks [14] is an extensible debugging tool for CORBA-based distributed object systems. It is comprised of an extensible analysis and visualization engine, and by a collection of probes, which are accompanied by a probe framework. Among other features, it is capable of replaying execution traces off-line. It provides a set of probes for monitoring common CORBA and distributed object application events, like client and servant pre-invoke and post-invoke, exchanged messages, request processing time, and others. Instrumentation may be automatic (as with the message exchange probes), or manual (for user-defined probes, as well as for some OVATION-provided probes). Unlike our tool, OVATION is a monitoring and analysis system. This means that it is not possible to use it to interact with the running distributed system, at least not with the richness attainable with a symbolic debugging tool. Also, as with all monitoring and analysis tools, instrumentation must be thought up-front. And finally, its probe framework (including instrumentation macros) is written in C++, which leads us to believe that, at the time of this writing, no other languages are supported for applications.

IBM's Object Level Trace: Object Level Trace (OLT)[6] is an extension to the IBM distributed debugger. Unlike other debugging tools targeted at distributed object systems, IBM's Object Level Trace incorporates a symbolic debugging service and, like our tool, it allows the user to follow the flow of control of his application from client to server, abstracting middleware details away. The concepts are similar, but OLT does not try to be a symbolic debugger – there are no explicit distributed threads, only message tracking. We are also not quite sure about how extensible OLT is, as it is a closed-source implementation. As far as the authors knowledge go, OLT is restricted to IBM's own technology, like WebSphere and the Component Broker. Also, neither OLT nor IBM's distributed debugger seem to be concerned with application instantiation, at least not beyond providing a simple facility for firing remote processes.

P2D2: The Parallel and Distributed Program Debugger[2] (P2D2) aimed at being a portable debugger for parallel and distributed (cluster) applications based on middleware such as PVM and MPI, as well as runtime environments like HPF. Our approach is based on many of the principles of P2D2, such as a decoupled client-server architecture, the use of a standard, language-independent wire protocol, and the leveraging of existing symbolic debuggers. The difference lays in the fact that we are counting on being able to adapt existing Eclipse-based symbolic debuggers so they can be integrated into our implementation, whereas P2D2 attempted to provide a standardized foundation all by itself. This means our implementation is much simpler. Also, P2D2 attempted to provide a debugger-neutral layer on top of existing symbolic debuggers at the server-side, meaning that all of its wire protocol is standardized. We adopt a different approach with our two-protocol local agent, again trying to facilitate reuse of existing Eclipse-based debugger clients. On the other hand, we require remote-debugging-enabled symbolic debugger clients. Regarding process management, P2D2 delegates the responsibility for process creation to the underlying infrastructure, whereas our implementation takes this responsibility upon itself. While this means we had to develop our own infrastructure, it also meant we did not have to think about interfacing with existing infrastructures.

5 Conclusions and Future Work

This paper presented a simple technique and an extensible Eclipse-based tool for symbolic debugging of distributed object applications. Our rationale for the development of this work has followed two principles: portability and usefulness. Our tool is portable because the tracking technique is simple, and based on elements that are common to synchronous-call middleware platforms. It is also portable because instrumentation requirements are not demanding, and because we can leverage existing, open source debugging clients. The conclusion that it is a suitable candidate for surviving technology evolution draws from these characteristics. Our tool is useful because it helps the user fight the maze effect by bringing debugger abstractions on par with middleware abstractions, because it helps detecting failures and distributed deadlocks, and also because it streamlines the workflow with its process management and instantiation infrastructure. There are many issues we did not attempt to address in this work, and which could be of value. Integration of more scalable visualization mechanisms (like event and call graphs), and automatic analysis tools [7] would be two examples. Addressing perturbations to the underlying execution with a replay facility would be another avenue. We are currently not, however, any close to having portable execution replay in multithreaded environments. A demonstration screencast, and the source code for our tool, can be obtained at http://god.incubadora.fapesp.br.

References

1. Apache BCEL website, `http://jakarta.apache.org/BCEL`
2. Cheng, D., Hood, R.: A portable debugger for parallel and distributed programs. In: Proc. of the 1994 ACM/IEEE conf. on Supercomputing, pp. 723–732 (1994)
3. Chiba, S.: A Metaobject Protocol for C++. In: Proc. of the ACM OOPSLA 1995, pp. 285–299. ACM Press, New York (1995)
4. Damodaran-Kamal, S.K.: Testing and Debugging Nondeterministic Message Passing Programs. PhD thesis, Univ. of Southwestern Louisiana (1994)
5. Gait, J.: The Probe Effect in Concurrent Programs. Soft.: P & E 16(3), 225–233 (1986)
6. IBM. Object Level Trace, `http://publib.boulder.ibm.com/infocenter/wasinfo/v4r0/topic/com.ibm.websphere.v4.doc/olt_content/olt/index.htm`
7. Kranzlmueller, D.: Event Graph Analysis for Debugging Massively Parallel Programs. PhD thesis, Johannes Kepler University, Linz, Austria (September 2000)
8. Krishnamurthy, Y., et al.: The Design and Implementation of Real-Time CORBA 2.0: Dynamic Scheduling in TAO. In: Proc. of 10th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 121–129. IEEE Computer Society Press, Los Alamitos (2004)
9. Li, J.: Monitoring and Characterization of Component-Based Systems with Global Causality Capture. In: Proc. of the 23rd ICDCS, pp. 422–433 (2003)
10. Lima, A., et al.: A case for event-driven distributed objects. In: Proc. of DOA 2006. LNCS, pp. 1705–1721. Springer, Heidelberg (2006)
11. Maes, P.: Concepts and experiments in computational reflection. In: Proc. of the OOPSLA 1987, pp. 147–155 (1987)
12. Mittal, N., Garg, V.K.: Debugging Distributed Programs Using Controlled Reexecution. In: Proc. of the 2000 ACM PODC, pp. 239–248. ACM Press, New York (2000)
13. Nelson, B.J.: Remote Procedure Call. PhD thesis, Carneggie Mellon University, Pittsburg, PA (1981)
14. OCI. OVATION Website, `http://www.ociweb.com/products/ovation`
15. Pancake, C.: Establishing Standards for HPC System Software Tools, `http://nhse.cs.rice.edu/NHSEreview/97-1.html`

16. Schwarz, R., Mattern, F.: Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail. Distributed Computing 7(3), 149–174 (1994)
17. Stallman, R.M.: GDB Manual: The GNU Source Level Debugger. FSF, Cambridge, Massachusetts (1987)
18. Sun Microsystems. The Java Platform Debug Architecture (2007), http://java.sun.com/prhttp://java.sun.com/products/jpda/index.jsp
19. Wright, D., Freeman-Benson, B.: How To Write an Eclipse Debugger, http://www.eclipse.org/articles/Article-Debugger/how-to.html

A Bluetooth-Based JXME Infrastructure*

Carlo Blundo and Emiliano De Cristofaro

Dipartimento di Informatica e Applicazioni, Università degli Studi di Salerno
Via Ponte Don Melillo - I-84084 Fisciano (SA), Italy
{carblu, emidec}@dia.unisa.it

Abstract. Over the last years, research efforts have led the way to embed computation into the environment. Much attention is drawn to technologies supporting dynamicity and mobility over small devices which can follow the user anytime, anywhere. The Bluetooth standard particularly fits this idea, by providing a versatile and flexible wireless network technology with low power consumption.

In this paper, we describe an implementation of a novel framework named **JXBT** (JXME over Bluetooth), which allows the JXME infrastructure to use Bluetooth as the communication channel. By exploiting the JXME functionalities we can overcome Bluetooth limitations, such as the maximum number of interconnectable devices (7 according to the Bluetooth standard) and the maximum transmission range (10 or 100 meters depending on the version). To test the lightness of **JXBT**, we designed and evaluated `BlueIRC`, an application running on top of **JXBT**. This application enables the set up of a chat among Bluetooth-enabled mobile devices, without requiring them to be within transmission range.

1 Introduction

In the last years, much effort has been placed on developing services for mobile devices. Smartphones are nowadays small and powerful enough to turn into fundamental working instruments. Technology advances also involved mobile communication technologies, as with the growth of Bluetooth [12]. This technology is a versatile and flexible short-range wireless network technology with low power consumption. It operates in a license-free frequency, so that user is not charged for accessing the network nor he needs an account with any company, thus allowing a relevant decrease of communication costs. Furthermore, it provides the possibility of automatically discovering other devices (and services exposed by them) within their communication range, thus allowing a dynamic set up of an ad-hoc network. For more details about this technology, we refer to the extended version of this paper [11].

Moreover, the evolution of smartphones drove researchers to develop collaborative protocols targeted to mobile devices. These protocols, can be used as the underlying technologies for complex distributed applications. In particular,

* This work has been partially supported by the European Commission through the IST program under contracts FP6-1596 (AEOLUS).

much attention has been placed on peer-to-peer protocols fitting mobile environments where devices have limited resources. In this paper, we focus on one of the most diffused Java-based peer-to-peer protocol, JXTA and in particular on its version for the mobile environment, JXME.

The JXTA technology is a set of open protocols that allows any connected device on the network to communicate and collaborate in a peer-to-peer style [8,22]. Its main features are: (i) interoperability, overcoming the problem of binding each peer-to-peer system to a single service and to a single infrastructure; (ii) platform and programming language independence; (iii) ubiquity, being implementable on every device ranging from mobile devices, PDAs, to PCs and servers. JXTA provides several features, such as: discovering other peers in the network; self set-up in peer groups; exposing and discovering network services; communicating in a direct way through *pipes*; monitoring other peers' activities.

The JXME project [10] is aimed to bring JXTA functionality to mobile devices with limited resources within the Java 2 Micro Edition. In fact, JXTA cannot be ported on the J2ME environment as it is. First, JXTA messages are XML-structured and require a parser, whose installation on a J2ME device is not straightforward. Then, maintenance of caches is too memory-consuming. Finally, JXTA peers are always in listening mode and this might not be achievable within J2ME environment.

Because of resource limitations, the JXME architecture heavily relies on JXTA *relays*. Relays are JXTA peers, which are able to manage pipes, advertisements, groups, routing. They act as access points to the JXTA network for mobile peers. These mobile peers communicate with the relay through a binary HTTP connection, using messages compliant to JXTA ones. Relay peers are in charge of creating resources, searching resources, messages sending and receiving. The JXME architecture is presented in Figure 1 [23]. This version of JXME is called Proxy-based, since a relay peer is needed to enter an existing JXTA network. On the other hand, the need of a proxyless version has become more and more insistent in order to achieve a full *ad-hoc* P2P scenario. However, to the best of our knowledge, no version of Proxyless JXME has been released for the J2ME Connected Device Limited Configuration (CLDC) [16], but only for the J2ME Connected Device Configuration (CDC) [16], thus restricting peers to run over relatively powerful devices, such as PDAs. In our work, we focus on CLDC in order to allow the creation of JXME networks among smartphones, which are

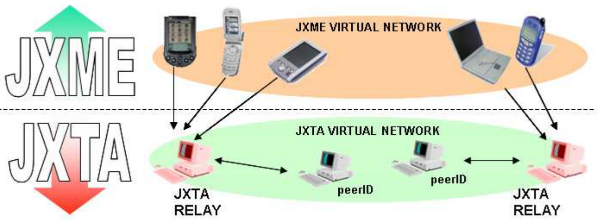


Fig. 1. The JXME Architecture

nowadays in widespread use. More details on the JXTA, J2ME, and JXME technologies can be found in the extended version of the paper [11].

From the synergy of the JXME and the Bluetooth technologies, we want to achieve a twofold goal. First, we want to provide JXME with the required support to use Bluetooth as the communication channel. Second, we want to exploit JXME functionalities in order to overcome Bluetooth limitations, such as the maximum number of interconnectable devices (7 according to the Bluetooth standard) and the maximum transmission range (10 or 100 meters depending on the version). Hence, we developed a novel framework, **JXBT** (JXME over Bluetooth). The JXBT infrastructure allows us to interconnect more piconets[1] in a transparent way. As a result, users can exploit all the classical Bluetooth features (such as file exchanging or chat services) without requiring peers to be within transmission range.

In order to provide a proof-of-concept for testing the lightness of **JXBT**, we designed and evaluated `BlueIRC`, an application running on top of **JXBT**. This application enables the set up of a chat among Bluetooth-enabled mobile devices, without requiring them to be within the range. More details are presented in Section 4. We evaluated the performances and usability of our framework on devices nowadays available on the market at average costs. Indeed, **JXBT** and `BlueIRC` have been tested on real mobile devices such as Nokia N70 and Nokia N73 smartphones. Our performance evaluation (presented in Section 5) shows that **JXBT** is efficient and stable. It can used as the underlying technology for developing complex and useful applications which can be released for final users owning standard Bluetooth-enabled mobile phones.

2 The JXBT Project

Bluetooth standard suffers of some constraints as maximum number of interconnectable devices (seven according to the Bluetooth standard) and limited transmission range (i.e., 10 or 100 meters). To this aim we provided the JXTA-JXME infrastructure with the Bluetooth's support as the communication channel enabling interactions over a P2P infrastructure. For this goal, we implemented a dedicated framework that we named **JXBT** (JXME over Bluetooth). Using **JXBT**, we are able to extend Bluetooth functionalities by providing a transparent middleware, which allows users to overcome the above mentioned limitations. Hence, users can exploit all the classical Bluetooth features (such as file exchanging or chat services) without requiring peers to be within the transmission range. Actually, the Bluetooth standard defines scatternets, i.e., sets of piconets connected through sharing devices, but it does not specify a scatternet formation algorithm. Although much effort is being put in the research of an optimal algorithm (see for instance [26], [20]), scatternet features are not yet well-developed in practice and to the best of our knowledge, there is no support for scatternet formation within J2ME.

[1] A piconet is the network formed by devices within each other's Bluetooth range.

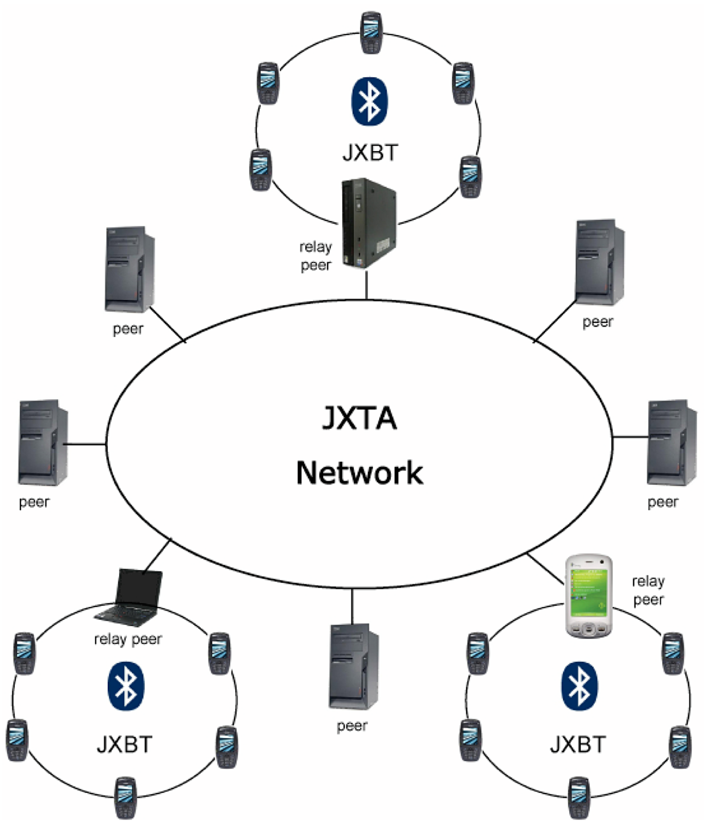


Fig. 2. The **JXBT** Framework Architecture

2.1 Design

The JXME technology allows to instaurate communications based on TCP and UDP, within the J2ME environment. While porting JXME to use Bluetooth as the communication channel, several issues have to be considered. TCP and UDP are widely supported for Bluetooth connections, but not within J2ME environment which is the programming environment of the JXTA/JXME framework. Furthermore, neither IP addresses nor domain names can be used to identify and address peers. As a consequence, we had to implement the required support for the dynamic discovering of peers and to handle point-to-point connections between peers and the relay. To this aim, we used the `JAWBT` API [7], defined by the JSR-82, to support Bluetooth connections in J2ME devices.

As shown in Figure 2, the **JXBT** framework architecture recalls the JXTA/JXME one. It contains the JXTA network and the relay peers, the JXME networks and the mobile peers. However, the Bluetooth layer has to be inserted in order to support transmissions. This layer lies both on the relay peer and on the mobile peer, it is implemented by a Bluetooth stub in a J2ME class using the JSR-82 API.

The physical communication is carried out by the Bluetooth stubs in the classical master/slave way. To this aim, two dedicated classes have been implemented, namely `BTMaster` and `BTSlave`. In our implementation, mobile peers act as masters (i.e., they generate and expose a Bluetooth service), while, relay peers act as slaves (i.e., they perform the Bluetooth inquiry and instaurate the connection). This choice is driven by efficiency: we tried to move the maximum amount of workload from mobile peers (which usually have limited resources) to the relay peers (which can be a standard non-mobile computer). Therefore, we let the relay peer periodically perform the inquiry to discover new peers or peers that have recovered after a fault. In fact, inquiry operation is a quite expensive task, especially if it has to be performed periodically. Furthermore, in order to forward messages, the relay peers has to know which mobile peers are within its range, and, for all of them, it has to keep track on the group to which they belong. The relay peer can read this information when discovering the mobile peer. Moreover, our choice supports peers' mobility: peers can move and change the reference relay peer and the infrastructure takes care of the synchronization. On the other hand, the choice of implementing the relay peer as master would slow down the computation, requiring relay peers to ask mobile peers to send a special message containing all the needed information (e.g., peer identity and/or group identity).

The main task of the **JXBT** framework is to provide to the core JXME class, `PeerNetwork`, the support for using Bluetooth as communication channel. To this aim, we had to implement an intermediate layer, represented by the `BTPeerNetwork` class. The resulting software structure is presented in Figure 3. The class `BTPeerNetwork` extends the `PeerNetwork` class in order to implement I/O operations on a Bluetooth channel, addressing either the `BTMaster` or the `BTSlave` class, according to the role of the peer. This strategy has been borrowed from Jadabs' implementation [4], more details can be found in Section 3. We have reused some utility classes of Jadabs to handle queues and to parse received messages. However, although a different thread was generated for each queue, no synchronization was provided in [4]. Therefore, we had to manage the synchronization of the multithreaded programming. Furthermore, we had to implement classes for groups and pipes management, as required by JXME: `PeerGroup` and `Pipe` (groups and pipes creation and searching), `PeerGroupHandle` and `PipeHandle` (management of queues for users joining some groups or listening on some pipes). These classes are needed to maximize the standardization and the compatibility with the JXME specification. In such way, the P2P developer is not required to know all implementation details about JXME requests addressed to the relay peer and about result retrieval.

We also remark that this design supports mobility because it allows a mobile peer to disconnect from a relay peer and re-connect to another. Pipes and groups memberships remain unaltered and allows the peer to recover messages not received while disconnected, thanks to the use of pipes and acknowledgments.

In our implementation, relay peers hold a pool of Bluetooth connections and periodically perform inquiry to check whether a new mobile peer has come within

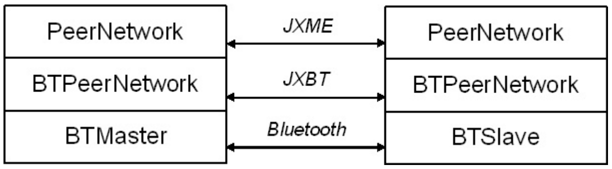


Fig. 3. The PeerNetwork abstraction over Bluetooth within **JXBT**

range. Whenever a mobile peer is discovered, the relay peer recovers and stores information about its identity and the group to which it belongs. Then, it sends to the new peer the state of available groups and pipes. In this way, mobile peers are not loaded by any computation: in order to use Bluetooth to communicate with other peers, mobile peers just have to generate a service and wait to be discovered. As for the relay peer, the Bluetooth stub is to be implemented not only to support Bluetooth interactions. In fact, it also has to provide a proxy mechanism to convert Bluetooth raw messages traveling in the **JXBT** environment into HTTP-formed messages traveling in JXTA network over HTTP and viceversa. Moreover, since for the JXTA network only the relay peer is connected, the relay peer is in charge of associating peers to pipes and groups.

We remark that after a mobile peer has been discovered, it is logically connected to other peers as they were within the Bluetooth transmission range, though belonging to other piconets. In fact, more **JXBT** networks are interconnected to each other through the use of JXTA, allowing mobile peers to ignore whether other peers are within transmission range (see Figure 2). Moreover, application developers that want to use **JXBT** do not have to take care of any implementation detail related to the use of the Bluetooth channel, since all the work is performed by our API. Programmers develop their application as for the standard JXME architecture but they can exploit the Bluetooth facilities in a transparent way. The only difference is to instantiate `BTPeerNetwork` objects, rather than `PeerNetwork` ones.

2.2 Mobile Peer Implementation

Whenever a mobile peer wants to enter a **JXBT** network, it has to initialize the connection, start a new thread, and perform the connection. The main classes needed to initialize the connection are:

1. `PeerNetwork`. When creating an instance of it, the user specifies the name of the peer, the group id, the inquiry maximum timeout, the URL, and whether the peer is the relay.
2. `BTPeerNetwork`. It is contained in the `PeerNetwork`. It uses the `Messenger` utility class to perform communications. The role of the peer (relay or mobile) is also to be specified together with the URL.

3. `BTEndPoint`. It is contained in the `BTPeerNetwork`, together with a connection pool and a message queue. It implements all the physical details of the Bluetooth communications. It is in charge of maintaining the Bluetooth connections, implementing the messenger mechanisms, and above all sending and receiving actual data.

Once the connection has been initialized, the main steps needed to create the connection are:

1. Adding a message listener on the Bluetooth channel, through the `BTPeerNetwork` object.
2. Performing the connection on the Bluetooth channel, through the `BTEndpoint` object.
3. Instantiating a `PeerGroup` object for the groups and a `Pipe` object for the pipes.

As a result, **JXBT** mobile peers have to run simple and short portions of code in order to connect themselves to the network. The Java fragment which has to be written is showed in Figure 4.

```
try {
  // open a service
  _service = (StreamConnectionNotifier) Connector.open(_serviceURL);
} catch (IOException e) {
  e.getLocalizedMessage();
}
while(true) {
  try {
    localDevice = LocalDevice.getLocalDevice();
    // accept connections
    StreamConnection conn = _service.acceptAndOpen();
    ...
    synchronized(this) {
         _newConnection = true;
         this.notifyAll();
    }
  }
}
```

Fig. 4. Java code for mobile peer service generation and to wait for discovering

2.3 Relay Peer Implementation

Whenever a relay peer is started, as for the mobile peer, it has to initialize the connection, start a new thread, and perform the connection in order to boot as JXME relay peer.

The main classes needed to initialize the connection from the relay side are:

1. `PeerNetwork`. When creating an instance of it, the user specifies the identity of the peer, the inquiry maximum timeout, the URL, and that the peer is going to be the relay.
2. `BTPeerNetwork`. As for mobile peers, it is contained in the `PeerNetwork` object and uses a `Messenger` utility class to perform communications. The role of the peer is to be specified, together with the URL.

```
public void connect() throws
 NoPeerAvailableException, IOException {
  // JXME peers discovering
  ...
  _agent.startInquiry(DiscoveryAgent.GIAC, this);
  // wait for discovering peers - service discover
  for(int index = 0; index<_deviceCounter; index++){
     ...
     int transactionId = _agent.searchServices
        (_attrSet, _uuids, _devices[index], this);
     ...
  }
  // open a connection on the discovered service
  for(index = 0; index < _serviceCounter; index++) {
     ...
     // try to connect
     try {
         ...
         StreamConnection conn = (StreamConnection)Connector.open(_services[
              index]);
         RemoteDevice remoteDevice = RemoteDevice.getRemoteDevice(conn);
         BTConnectionHandle handle = new BTConnectionHandle(conn,
              remoteDevice);
         _connectionPool.addConnection(handle);
         ...
     }
  }
}
```

Fig. 5. Java code for connection of a relay peer to a discovered mobile peer

3. `BTEndPoint`. As for mobile peers, it is contained in the `BTPeerNetwork` object. This object handles all details about Bluetooth communication.

The main steps carried out during the connection creation are:

1. Adding a message listener on the Bluetooth channel, through the `BTPeerNetwork` object.
2. Performing the connection on the Bluetooth channel, through the `BTEndpoint` object.
3. Instantiating a `PeerGroup` object for the groups and a `Pipe` object for the pipes.
4. Setting the peer ID.

As a result, the relay peer has to inquiry and connect to available mobile peers. This operation is straightforward, thanks to the Bluetooth standard and the JSR-82 API. The Java code showed in Figure 5 is devoted to such task.

More details about our implementation can be found in the extended version of this paper [11].

3 Related Work

Both P2P and ubiquitous computing are actual topics in the scientific community. Therefore, researchers have put a lot of effort on designing P2P protocols which are suitable for mobile wireless devices.

In [14] it has been presented Jadabs, a dynamic lightweight architecture for resource constrained device, Jadabs allows to build a distributed peer to peer infrastructure similar to JXTA. However, Jadabs cannot run on CLDC/MIDP because of the limitations imposed by CLDC (reflection and dynamic class loading are not supported). Therefore, the Jadabs-CLDC project [3] was carried out in order to overcome this limit. Such a project is based on the synergy between Jadabs and JXME. The project Jadabs-JXME-BT [4] was aimed to provide Bluetooth support to Jadabs-CLDC. However, this project cannot be considered as a real implementation of JXME over Bluetooth. In fact, Jadabs-JXME-BT only implements the JXME messaging system, it does not provide mechanisms to create and handle pipes and groups. Finally, it lacks of the management of modules and advertisement. As a result, Jadabs-JXME-BT does not provide a full JXME-compliant interface.

Other research is targeted to achieve a synergy between MANET and P2P in mobile environment. For instance, PROEM [17] is a mobile middleware for ad-hoc networks based on WLAN within J2ME. Mobile Chedar [19] is similar to PROEM, but it provides support for peers with fixed P2P network connections. It uses Bluetooth as the underlying communication channel for supporting Chedar P2P protocol (see [18]).

The Java Community Process released the JSR-259 Ad-Hoc Networking API [5] to support communication between nodes in an ad-hoc network implemented in J2ME, allowing developers to deploy P2P application over mobile phones. However, no implementation is available yet.

Other interesting works are targeted to cooperative peer-to-peer applications running on mobile phones. For instance, issues related to development of peer-to-peer games in J2ME using Bluetooth has been deeply investigated in [24]. A peer-to-peer framework to support rapid development of mobile collaborative applications has been presented in [25]. It uses Bluetooth as the communication channel. Both papers rely on the P2P framework Peer2ME [21] which is targeted to Bluetooth-enabled mobile devices.

However, none of the cited projects achieves our goal of providing JXME with the required support to use Bluetooth as the communication channel. The

Properties	**Peer2 ME**	**Mobile Chedar**	**Jadabs-JXME-BT**	**JXBT**
Lightweight	Yes	Yes	Yes	Yes
Full compliance to JXME	No	No	No	Yes
Support for groups	Yes	No	No	Yes
Support for pipes	No	No	No	Yes
Support for synchronized multithreaded queues	No	No	No	Yes
Open-Source	No	No	Yes	Yes
Based on standard protocol	No	No	Yes	Yes
Overcome piconet range constrain	No	Yes	Yes	Yes

Fig. 6. Differences between **JXBT** and related projects

JXBT framework fulfills our requirements. In Figure 6, we summarize the main differences among **JXBT** and some related projects.

4 The BlueIRC Application

In order to provide a proof-of-concept of the applicability of our framework in a real world scenario, we developed an application, named BlueIRC working on top of the **JXBT** framework. BlueIRC is essentially an application enabling the creation of a chat among Bluetooth-enabled mobile devices, without requiring them to be within Bluetooth's transmission range. This application has been tested both in a simulation environment and by using real smartphones.

As we have shown in Figure 2, peers within the Bluetooth's range (piconet) are interconnected through the **JXBT** infrastructure. Then, relay peers provide access to the JXTA networks, thus allowing piconets to interconnect to each other. In this way, mobile devices network using Bluetooth are not bounded to the range of the Bluetooth layer. Furthermore, peers do not even need to know whether interacting peers are effectively within their transmission range. Figure 7 gives an overview of how BlueIRC works; Message 1 is exchanged within the same JXME network; while, Message 2 is handled by the framework and it is sent in a transparent way to a peer belonging to a different JXME network. As we can see in Figure 7, even when two mobile peers are in the same piconet, messages have to be processed by the relay peer. This limitation is due to the fact that the JXME proxyless version, as yet, only supports devices using CDC,

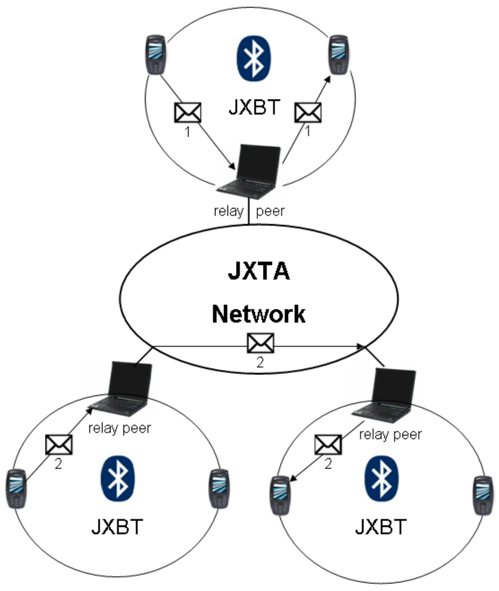


Fig. 7. How BlueIRC works

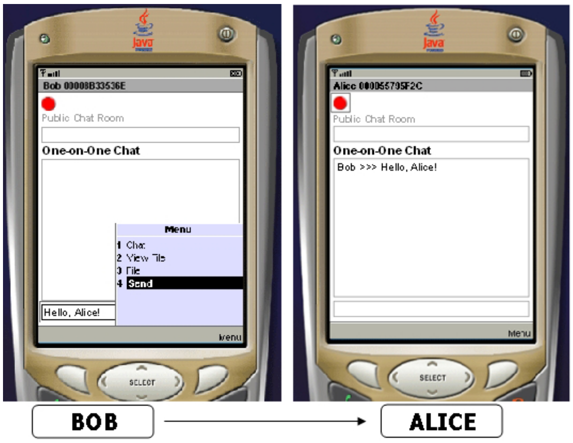


Fig. 8. `BlueIRC` in the *One-on-One Chat* modality

not CLDC, eliminating mobile phones. However, as soon as an implementation will be released, we plan to release the updated version of **JXBT**.

The `BlueIRC` chat provides two different chat modalities: the *One-on-One Chat* and the *Public Chat Room*. In the first case the user interacts with a single user of a group; while, in the second case, the user interacts with all the users of a group. The `BlueIRC` application provides users with several features, such as: selection of groups, selection of the chat modality (single or multicast), sending and receiving of messages and/or files, management of files and directory, contacts, and personal agenda.

Once the `BlueIRC` application has been started, the user has to confirm the connection to the system. As stated in Section 2, any peer entering the networks has to wait to be discovered from the JXTA relay peer. At this stage, the relay peer contacts via Bluetooth the new peer, which sends an identifying JXME message and gets the list of available groups. Now, the peer is connected to the JXTA network and he can choose a group sending a request to join it. As a response, it gets the list of all the peers in the group, both the ones in the same piconet and the ones reachable through the JXTA middle layer. At this point, the user can exploit all the `BlueIRC`'s features. As an example, the Figure 8 shows a screenshot of `BlueIRC` running in the *One-on-One Chat* modality. Whenever a user sends a message, a JXME message is sent to the relay peer. This checks whether the receiver is in the list of local (in-the-range) mobile devices. In this case, it delivers the message (see Message 1 of Figure 7), otherwise it has to route the message in the JXTA network over a pipe, so that the correspondent relay peer can appropriately deliver the message (see Message 2 of Figure 7).

Figure 9 shows an example of the *Public Chat Room* modality. The relay peer receiving the message broadcasts the message to local (in-the-range) peers of the group. It also sends the message on the JXTA network over the pipe to allow other relay peers to spread the message to other peers of the group. `BlueIRC`

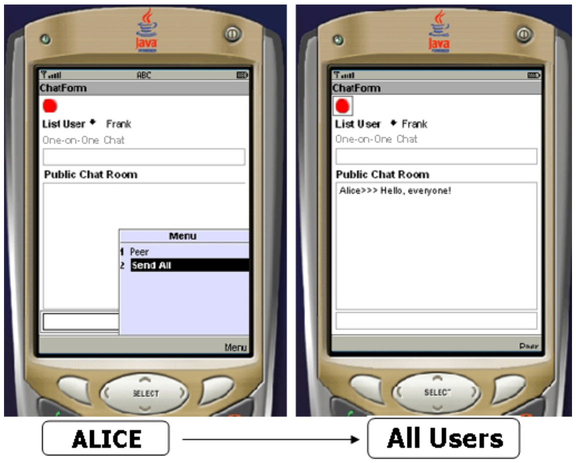


Fig. 9. BlueIRC in the *Public Chat Room* modality

is designed to handle the two chat modalities simultaneously allowing users to easily switch between them. In fact, the BlueIRC user interface presents two windows, a bigger one with the active chat modality and a smaller one for the not active one. Moreover, if the application is in the *One-on-One Chat* modality and a broadcasted message is received (i.e., the message is sent in the *Public Chat Room* modality), then it will not be discarded but it will be prompted in the smaller window. The same happens in the opposite situation.

Using the BlueIRC application, users can send and receive files. In fact, it has been implemented the support to access and manage devices' filesystem. Users can also browse files, see and edit their properties, or delete them. This feature was realized through the use of the JSR-75 API [6], which nowadays it is supported by almost all J2ME-powered smartphones.

5 Performance Evaluation

In this section, we analyze the performance of **JXBT** and BlueIRC in order to evaluate their lightness and usability in real world scenarios. To this aim we set up the following test bed:

- JXTA network: 2 PCs IBM ThinkCentre 50, Pentium 4 2,6 GHz with 760 MB RAM, running Windows XP Professional SP 2, acting as relay peers for the JXME networks. The two PC were equipped with Bluetooth TrendNet TBW-102UB USB dongles. We used the BlueCove [2] implementation of the JSR-82 Bluetooth API for J2ME.
- Peer 1: Nokia N73 mobile phone running Symbian OS 9.1, compliant with MIDP 2.0, JSR-82, and JSR-75 standards.
- Peer 2: Nokia N70 mobile phone running Symbian OS 8.1a, compliant with MIDP 2.0 and JSR-82 standards.

The first test was run to evaluate the overhead taken by our framework to send a file within the BlueIRC application. We have compared times to send files for the following applications:

1. BlueIRC in *One-on-One* modality.
2. A simple J2ME midlet sending files over Bluetooth.

Actually, only using BlueIRC it is possible to exchange files over Bluetooth without requiring devices to be within Bluetooth transmission range. However, we wanted to evaluate the overhead required by the **JXBT** infrastructure, so we restricted this test to a scenario where two devices were within range and were using the same relay peer to enter the JXTA/JXME network. We remark that **JXBT** suffers from the limitation of requiring all communications to be routed through the relay peer. However, this limitation is related to the lack of a proxyless JXME implementation for the CLDC. As soon as it will be released, we plan to update **JXBT** so that it will use the proxyless JXME rather than the proxy-based version. Figure 10 shows times required to send files of size ranging from 1 KB to 55 KB with an incremental step of 1 KB. Times show that BlueIRC performs around 2.2 times slower than the simple J2ME application which exchanges data directly without the use of an intermediate device (relay peer). However, to use J2ME, devices must be within Bluetooth transmission

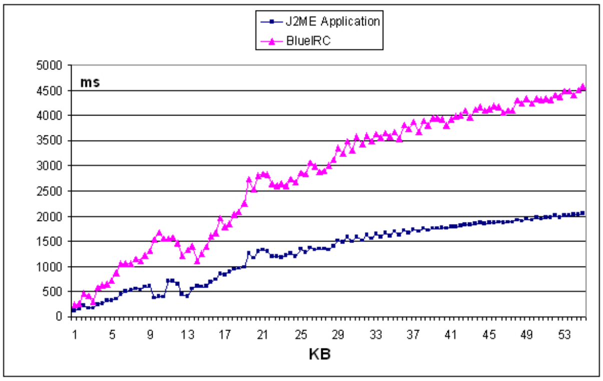


Fig. 10. Times for sending a file

range; while, in our framework this is not required. With the introduction of a proxyless version of JXME for CLDC, the differences in Figure 10 will be drastically reduced. The 2.2 factor is what we have to pay to overcome the Bluetooth transmission range limitation. In order to allow two Bluetooth device to communicate even though they are not within transmission range, we should add to times of Figure 10 the delay of the JXTA network, but this is independent from our **JXBT** framework. All JXTA-based applications incur in such a delay. Actually, several works evaluating JXTA performance have been published during last years, such as [15], [13], and [9].

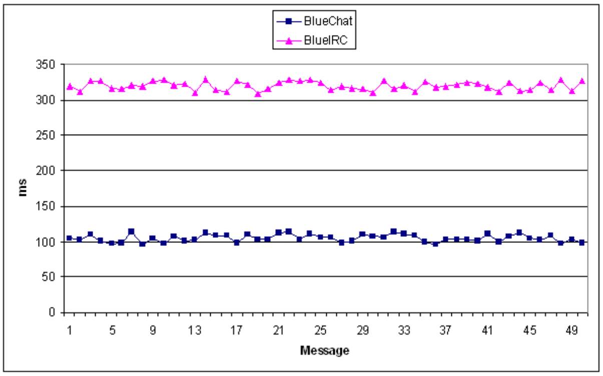


Fig. 11. Times for sending a message

The second test was aimed to evaluate the efficiency in exchanging messages within chat applications. We have compared times to exchange messages for the following applications:

1. `BlueIRC` in *One-on-One* modality.
2. BlueChat [1], the most famous chat application for J2ME and Bluetooth-enabled mobile phones.

BlueChat does require two devices to be within Bluetooth transmission range to exchange a message, while `BlueIRC` does not. As for the first test, also in this test we do not consider the transmission delay induced by the JXTA framework. Figure 11 shows times required to send a simple HELLO message. Tests were repeated 50 times in order to compute a significant average. Times show `BlueIRC` performs around 3 times slower than BlueChat, which exchanges data directly without the use of an intermediate device (relay peer).

6 Conclusions and Future Works

In this paper, we have presented **JXBT** a Bluetooth-based implementation for a JXME infrastructure. We also proposed a useful chat application, namely `BlueIRC`, running on top of **JXBT**. We notice that using **JXBT** results in a performance degradation compared to other available chat applications. But, this is due to the JXME proxyless constrain for CLDC that, instead of allowing devices to communicate directly, forces to route all communications through the relay peer. We will easily overcome this limitation as soon as a proxyless implementation of JXME for the CLDC will be released. We remark that, only using **JXBT** users can interconnect more piconets and exploit all Bluetooth features beyond Bluetooth transmission range. It would be possible to interconnect more piconets to form a scatternet. But, to our knowledge, only theoretical results are available, no real implementation has been deployed yet. In Figure 12, we

Limitation	Midlet	Blue Chat	BlueIrc	Solution
Textual message exchange	Yes	Yes	No	None
No broadcast/groups support	Yes	Yes	No	None
Communication within piconet range	Yes	Yes	No	Use scatternets if a formation algorithm is released for J2ME
Message routed through a relay peer	No	No	Yes	Update **JXBT** with J2ME proxyless version for CLDC

Fig. 12. Limitations of analyzed applications and possible solutions

summarized the limitations of the three applications we analyzed in this paper as well as possible solutions.

As future works, we will release a new version of **JXBT** as soon as the JXME proxyless version for CLDC will be available. Moreover, we plan to set up a simulation environment in order to test the framework scalability both in the number of supported relay peers and in the number of connected mobile peers. We also want to tune all parameters and timeouts in order to improve performance. Finally, we would deploy more mobile peers in order to compare simulation results against real world devices in noisy or crowded environments.

References

1. BlueChat, http://www.getjar.com/products/7545/BluetoothChat
2. BlueCove, http://sourceforge.net/projects/bluecove/
3. Jadabs-CLDC, http://jadabs.berlios.de/jadabs-cldc/
4. Jadabs-JXME-BT, http://jadabs.berlios.de/jadabs-cldc/multiproject/jxme-bt-j2me/
5. JSR 259: Ad-Hoc Networking API, http://jcp.org/en/jsr/detail?id=259
6. JSR 75: PDA Optional Package for the J2ME Platform, http://jcp.org/en/jsr/detail?id=75
7. JSR 82: Java APIs for Bluetooth, http://www.jcp.org/en/jsr/detail?id=82
8. The JXTA Project, http://www.jxta.org
9. Antoniu, G., Hatcher, P., Jan, M., Noblet, D.A.: Performance evaluation of JXTA communication layers. In: CCGrid 2005. Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid, vol. 1, pp. 251–258 (2005)
10. Arora, A., Haywood, C., Pabla, K.S.: JXTA for J2ME – Extending the Reach of Wireless With JXTA Technology. In: JavaOne Conference (2002)
11. Blundo, C., Cristofaro, E.D.: JXBT: JXME over Bluetooth. Technical report, Università di Salerno, http://www.dia.unisa.it/dottorandi/emidec/JXBT-Extended.pdf
12. Chatschik, B.: An overview of the Bluetooth wireless technology. IEEE Communication Magazine 39, 86–94 (2001)
13. Dai, Z., Fang, Z., Han, X., Xu, F., Yang, H.: Performance Evaluation of JXTA Based P2P Distributed Computing System. In: CIC 2006. Proceedings of the 15th International Conference on Computing, pp. 391–398 (2006)

14. Frei, A., Alonso, G.: A dynamic lightweight Platform for Ad-Hoc Infrastructures. In: PerCom 2005. Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications, pp. 372–382. IEEE Computer Society Press, Los Alamitos (2005)
15. Halepovic, E., Deters, R.: The jxta performance model and evaluation. Future Gener. Comput. Syst. 21(3), 377–390 (2005)
16. Keogh, J.E.: J2ME: The Complete Reference. McGraw-Hill, New York (2003)
17. Kortuem, G.: PROEM: A Middleware Platform for Mobile Peer-to-Peer computing. SIGMOBILE Mob. Comput. Commun. Rev. 6(4), 62–64 (2002)
18. Kotilainen, N., Vapa, M., Weber, M., Töyrylä, J., Vuori, J.: P2PDisCo - Java Distributed Computing for Workstations Using Chedar Peer-to-Peer Middleware. In: IPDPS 2005. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, pp. 182–185 (2005)
19. Kotilainen, N., Weber, M., Vapa, M., Vuori, J.: Mobile Chedar - A Peer-to-Peer Middleware for Mobile Devices. In: PERCOMW 2005. Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops, pp. 86–90 (2005)
20. Law, C., Mehta, A.K., Siu, K.-Y.: A new Bluetooth scatternet formation protocol. Mobile Networks and Applications 8(5), 485–498 (2003)
21. Lund, C.-H.W., Norum, M.S.: The Peer2Me Framework - A Framework for Mobile Collaboration on Mobile Phones. Master's thesis, Department of Computer and Information Science - Norwegian University of Science and Technology (2005)
22. Schollmeier, R.: A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In: P2P 2001. Proceedings of the First International Conference on Peer-to-Peer Computing, pp. 101–102. IEEE Computer Society Press, Los Alamitos (2001)
23. Tomarchio, O.: Progetto IS-MANET: JXTA Middleware for Mobile Ad-Hoc Networks. Technical report, http://zeus.elet.polimi.it/is-manet/Documenti/bo20040721-diit.ppt
24. Wang, A.I., Norum, M.S., Lund, C.-H.W.: Issues related to development of wireless peer-to-peer games. In: AICT-ICIW 2006. Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, pp. 115–120 (2006)
25. Wang, A.I., Norum, M.S., Lund, C.-H.W.: A peer-to-peer framework for mobile collaboration. In: SEA 2006. Proceedings of the 10th IASTED International Conference on Software Engineering and Applications (2006)
26. Zaruba, G.V., Basagni, S., Chlamtac, I.: Bluetrees-Scatternet Formation to Enable Bluetooth-Based Ad Hoc Networks. In: ICC2001. Proceedings of the IEEE International Conference on Communications, vol. 1, pp. 273–277 (2002)

Agreements and Policies in Cooperative Mobile Agents: Formalization and Implementation

Fuyuki Ishikawa, Nobukazu Yoshioka, and Shinichi Honiden

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan

Abstract. Organization of mobile agents into a group has appeared as a new paradigm for dynamic deployment of composite services. However, it has not been discussed how multiple mobile agents cooperate with each other, handling conflicts in their requirements. In response to this problem, this study proposes a model for cooperative mobility based on the notion of agreements. Agent behavior defined in the proposed model involves agreement establishment and enforcement for cooperative mobility. Such behavior can be customized only by specifying requirements/constraints of each agent, eliminating the necessity to write down the whole behavior to handle agreements. The model is described in a formal way, using Event Calculus, and it is proved the model leads to no occurrence of defined inconsistency. The model has been implemented on an existing agent framework, Freedia, combined with its dynamic partner management mechanism.

1 Introduction

The mobile-agent paradigm has opened up great possibilities for new applications [1,2]. A mobile agent is a software component that has the ability to move from one host to another with its state, allowing for local interaction with distributed components and/or selection of resources for its use. Mobile agents have therefore been utilized in various fields such as information retrieval, mobile computing, and dynamic resource allocation. In most use cases, agent migration is driven by each agent's requirements, and agents do not recognize migration behavior of other agents. Recently, several studies have proposed new models for *cooperative migration* where a group of mobile agents migrate together keeping locality with each other [3,4,5]. These models allow for dynamic delivery and deployment of composite services provided through combination of multiple agents. Cooperative migration has been used for composition of multimedia contents managed by agents [6,7,8] and for desktop transportation [9]. Cooperative mobility has been implemented by embedding commands like "*follow AgentA*", in agent behavior description, indicating the agent starts to follow migration of *AgentA*. The runtime platform then activates migration accordingly upon migration of *AgentA* (the followed one). Multiple agents are thus connected by "follow" relationships, leading to composite agents that migrate together.

However, such *follow* commands are included only in follower agents and just activate their following migration, without regard to the intention or state of the counterparts (followed agents). The implementation model thus assumes incorporation of interaction behaviors among agents for decision and mutual understanding between agents on how cooperative mobility is conducted. Such behaviors are necessary for effective cooperative behavior of each agent to fulfill requirements of other agents as well as its own. On the other hand, it is a heavy burden for each application developer to design and implement some global protocols and each agent's behavior to exchange requirements on mobility, determine and agree how to cooperate, and act accordingly. In addition, the resulting agent behavior also can be very complex, with behavior for cooperative mobility woven discretely into the application logic. The lack of support for decision and mutual understanding processes thus limits broader adoption of the emerging models of cooperative mobility, especially to service-oriented computing where agents discover and cooperative with each other at runtime [10]. Our objective in this study is thus to provide patterns of cooperative mobility including decision and mutual understanding processes and facilitate their implementation.

In general, the notion of agreements has been used to denote mutual understanding of interaction participants [11,12]. Agreements are established through explicit exchange of requirements of each participant in order to determine behavior of each participant that satisfies all of their requirements. However, it is heavy burden for application developers to specify behavior to establish agreements and act in complying with them. Especially, there is a significant issue in handling agreements, that is, assurance of consistency between terms in an agreement, between an agreement and behavior of its participant, and between agreements of an agent with different partners.

In response to these problems, this study proposes a model for cooperative mobility based on the notion of agreements. Agent behavior defined in the proposed model involves agreement establishment and enforcement for cooperative mobility. Such behavior can be customized only by specifying requirements and constraints of agents, eliminating the necessity to write down the whole behavior. The model is described in a formal way, using Event Calculus, and it is proved the model leads to no occurrence of defined inconsistency.

The proposed model has been implemented in an agent framework, Freedia [7,8], for development of agents that discover and cooperate with each other possibly while conducting cooperative migration. Besides common functionalities for agent execution and management, the Freedia framework provides mechanism for dynamic partner management [13]. The mechanism is combined with the proposed mechanism for cooperative mobility to handle failure in cooperative mobility that may lead to rebinding of partners.

2 Background and Motivation

2.1 Cooperative Mobility

In most mobile agent systems, each agent determines migration strategies (i.e., when and to where to migrate) according to its own requirements, not recognizing migration of other agents. Recently, different models have been proposed for cooperative migration of multiple mobile agents.

A model of *hierarchical mobile agents* first discussed cooperative mobility of multiple agents [3,4]. The model introduced *inter-agent migration* where an agent becomes contained by another agent and then carried by the containing agent. Figure 1 illustrates the model. *Agent2* migrates into and interacts with *Agent1* (Step 1, 2 in the figure). When *Agent1* migrates into a host or an agent, *Agent2* migrates to the same host (Step 3, 4). In this way, the model leads to a "mobile composite agent" that consists of multiple agents migrating together as a unit. Existing studies have provided frameworks to implement this model by using a command to "*enter into*" an agent.

One of attractive applications of cooperative migration is composition of multimedia contents managed by agents [6,7,8]. Services providing multimedia contents with related functions are composed by combining existing contents and functions. Each component content or function is encapsulated and managed by agents so that its provision is controlled according to the provider's policy. Upon such combination, mobility can be used to reduce communication cost, avoiding exchange of large data between distributed agents. Cooperative mobility allows handling the mobile composite service that consists of multiple mobile agents.

Although the metaphor of inter-agent migration is intuitive and often used to explain the model, the essence of the hierarchical model is establishment of a logical relationship where one agent keeps following another's migration. The relationship is very strong as a group of agents always migrates to the same host, which makes it difficult to satisfy host requirements of all the group members. A model of *loosed cooperative mobility* has proposed to relax the constraint [5]. In the model, an agent follows another's migration similarly, however, the migration target need not to be exactly the same. Instead, agents in a group migrate to hosts *located nearby* in some definition, for example, hosts in the same physical domain defined by the platform [14]. In Figure 2, *Agent2* follows migration of *Agent1*, migrating to a host in the same domain as *Agent1*. This loosed model of cooperative mobility is generalization of the hierarchical model, and still keeps

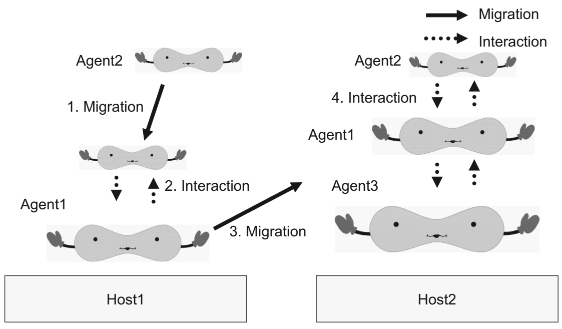


Fig. 1. Hierarchical Cooperation Model for Mobile Agents

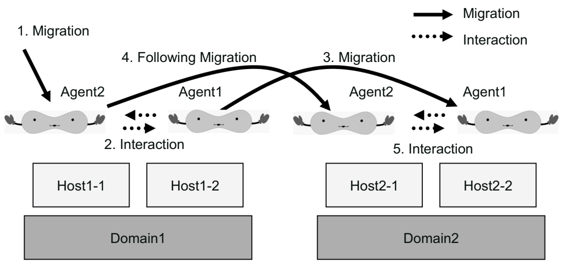


Fig. 2. Loosed Cooperation Model for Mobile Agents

locality of agents, enabling to reduce communication costs in long-running interaction. Existing studies have considered to implement this model by using a command to "*follow*" an agent.

2.2 Problems

Existing platforms for cooperative mobility have provided commands for agents to follow (or enter into) another agent. Although such commands have enabled cooperative mobility, they are too primitive in a sense decision and mutual understanding processes are not supported, especially for cooperation of agents provided by different organizations. It is necessary to facilitate to introduce agent behaviors to exchange requirements on mobility, determine and agree how to cooperate, and act accordingly. Such behaviors are often considered with a notion of agreements/contracts. In our context of cooperative mobility, agreements should be introduced so that:

– A follower agent, which delegates decision of migration to another, can be sure that its requirements/constraints on migration target hosts are satisfied.
– A followed agent, which often wants agents providing component services to follow it, can be sure that they actuality follow its migration.
– Agents can understand how to handle failures in cooperative migration.

When agreement-based cooperative behavior is developed, it is generally necessary to determine how to denote agreements, how to establish agreements between agents, and how to act in complying with established agreements (possibly with monitoring of counterparts). As this study focuses on a specific domain, that is, cooperative mobility, this study aims to facilitate such development by defining vocabularies and behaviors specific to the domain, as follows.

Vocabulary. Agreement vocabulary is defined so that implementation of typical patterns in cooperative mobility is facilitated, e.g., handling failure in migration.

Establishment. Vocabulary for requirements/constraints (or *policies*) exchanged to establish agreements is defined together with their matching mechanism so that agent behavior to establish agreements is implemented only by specifying requirements/constraints.

Enforcement. Runtime platform is provided that enforces established agreements so that behavior to comply with agreements is conducted properly with little implementation effort of developers.

In addition, consistency is one of the most significant properties upon implementing agreement-based cooperation. Consistency between defined behavior of an agent and its agreement should be ensured. For example, there should not be a situation where an agent prohibited from migration, the agent requires to achieve its goals. Similarly, consistency between multiple agreements an agent establishes with different partners should be ensured. For example, an agent should not agree to follow multiple agents as it is generally impossible.

In this study, semantics of the proposed agreements and related agent behaviors are defined in a formal way (using Event Calculus [15]) so that it can be proved that there can be no occurrence of defined types of inconsistency.

3 Agreements on Cooperative Mobility

This section describes a model for agreements regarding cooperative mobility as well as policies, requirements/constraints exchanged to establish agreements. In other words, this section describes the proposed global protocols or possible patterns of cooperative mobility.

3.1 Approach

In general, permission and obligation have been handled as typical constraints (policies or agreements) in distributed systems, and often expressed in a formal model of Event Calculus [16,19,20]. Event Calculus is a well-known formal notation for expression and reasoning about effects of actions [15]. Event Calculus considers predicates called *fluents* whose boolean value can change as effects of actions or time passage, and provides definitions and axioms to specify and reason about how fluent values change. For example, Initiate(*act1*, *flu*, *t*) denotes the fact occurrence of action *act1* at time *t* makes fluent *flu* to start to hold (such declaration often accompanies a universal quantifier on time, i.e., for all time *t*). With such notations, it is possible to specify how permissions and obligations are initiated or terminated upon occurrence of various actions or events.

This study follows such general approaches and introduces obligation of following migration, initiated upon migration of the followed agent, in order to denote cooperative mobility. Such obligation and prohibition may conflict with each other when they are given to one agent by multiple agreements, each of which thus should be carefully analyzed. This study therefore expresses cooperative migration as combination of permission (prohibition) and obligation.

Due to the space limitation, the proposed agreement model are described informally in this section, and only an example of the formal notations is described in Appendix A.

```
<HostAgreement agent="a1" host="h1">
    <Timeout>tout_h</Timeout>
    <Resources>hCond1</Resources>
    <TerminateEvents>ev</TerminateEvents>
</HostAgreement>
```

Fig. 3. Agreement for Host Usage

3.2 Agreements for Host Usage

Agreement Specification. Figure 3 shows an XML expression of agreements between hosts and agents, containing the following variables.

$a1$, $h1$ Denote agent and host that participate in the agreement, respectively.
$tout_h$ Denotes time span within which $a1$ has to migrate to $h1$. If $a1$ does not migrate within this time span, the agreement is discarded.
$hCond1$ Denotes resources provided to $a1$ by $h1$. Actual expressions include a set of descriptions of each resource such as CPU. Detailed notations are not given here as they are very general and implementation-dependent.
ev Denotes a set of events that lead to obligation on $a1$ to exit of $h1$. Such events can be defined by application developers, and may include various events such as security incidents.

Establishment of an agreement initiates permission and obligation to enter a host. The permission and obligation are terminated upon successful migration or discard of the agreement. When the agreement is effective, specified resources are provided and specified events lead to obligation to exit of the host. When initial migration is timed out or the agent exits of the host, the agreement is discarded.

Agreement Establishment. To establish agreements, an agent and a host exchange requirements and constraints, or policies, of each other. Currently, only policies on host resources are exchanged and matched. Events leading to obligation of outgoing migration are determined by each host, and timeout span is defined as a constant value.

An agent requests an agreement by sending resource conditions it requires to a host. Resource conditions consist of conjunction of inequality conditions on each resource, such as CPU, memory, and so on. On the other hand, each host has its acceptable conditions. If the host accepts the requirements sent by the agent, an agreement is established between them.

3.3 Agreements for Cooperative Mobility

Agreement Specification. In agreements between agents, for cooperative mobility, it is specified whether cooperative migration is conducted or not, and if so which agent follows the other. It is also necessary to include conditions for targethosts and behavior to handle faulty situations. Cooperative mobility considers migration of agents to hosts that are located "locally". The meaning of

```
<MobilityAgreement agent=”a1 a2”>
    <FollowType>type</FollowType>
    <Follower>follower</Follower>
    <FollowerHostConstraints>hCond1</FollowerHostConstraints>
    <Timeout>tout_a</Timeout>
</MobilityAgreement>
```

Fig. 4. Agreement for Cooperative Migration

“being local” depends on the implementation, and here is not limited to a specific definition.

Figure 4 shows XML expressions of agreements for cooperative mobility, containing the following variables.

$a1$**,** $a2$ Denote agents that participate in the agreement.

type Denotes type of cooperative migration and is any of *strong*, *weak*, or *none*. When the value is *strong* or *weak*, cooperative migration is conducted. Cooperative migration can fail, when $a1$ can not follow migration of $a2$ due to lack of adequate hosts. In such a case, the agreement is discarded if the value of *type* is *strong*. Interaction is continued without locality if the value of *type* is *weak*. When the value is *none*, cooperative migration is not conducted.

$follower$ Denotes the agent that becomes the follower ($a1$ or $a2$) when cooperative migration is conducted.

$hCond1$ Denotes host requirements of the follower, $a1$.

$tout_a$ Denotes time span within which $a1$ has to migrate following migration of $a2$ (when *type* is *strong* or *weak*).

Suppose *type* is not *none* and $a1$ is the follower. In the following situations, $a1$ becomes obliged to move to a host that is local to the current host of $h2$.

– When an agreement is established and $a1$ and $a2$ is not nearby.
– When migration of $a2$ occurs and locality becomes lost.
– When $a1$ has to move to another host due to forced-out events in the agreement with the current host.

When $a1$ is imposed such obligation on, it should try to find and migrate to a host that satisfies the locality condition to $a2$ as well as its own resource requirements. Implementation of such enforcement behavior will be described later in Section 4. If such following migration is not conducted within $tout_a$, the agreement is discarded when *type* is *strong*. After such migration is successfully conducted, $a1$ becomes inhibited to try to migrate to another host that is not local to the host of $h2$. When an agreement is finally discarded, given obligation and prohibition are terminated. Concrete description of these semantics, in Event Calculus, is shown as an example in Appendix A.

Agreement Establishment. To establish agreements for cooperative mobility, agents exchange requirements and constraints, or policies, of each other, as

Table 1. Policy Matching for Cooperative Mobility

*nearby*1, *nearby*2	*canfollow*1	*canfollow*2	*locality*	*follower*
strong is committed at least by one agent	(any)	*yes*	*strong*	*a*2
	yes	*no*	*strong*	*a*1
strong is not committed by any agent, and *weak* is at least by one agent	(any)	*yes*	*weak*	*a*2
	yes	*no*	*weak*	*a*1
none is committed by both agents	(any)	(any)	*none*	unnecessary
strong or *weak* is committed at least by one agent	*no*	*no*	— (fail)	

in agreements between agents and hosts. This exchange is actually part of interaction to establish a partnership between agents. Agents may exchange other application-level requirements or negotiate with each other. Here only interaction for cooperative mobility is described, which is merged into more general binding behavior.

Requirements and constraints each agent has contains the following items.

nearby Denotes which kind of cooperative migration is preferred. Value of this element is any of *strong*, *weak*, or *none*, as in the *type* element in agent agreements. *type* element in the established agreement becomes the "stronger" one of values given by the two agents. If at least one of the agents requires *strong*, the resulting *type* is *strong*. If any does not require *strong* and at least one requires *weak*, *type* is *weak*. If both require *none*, *type* is *none*

canfollow Denotes whether the agent can follow migration of the other or not. Value of this element is *yes* or *no*. This element does not affect if *type* is *none*. If both the agents have *yes*, the agent that waited for and received a request becomes the follower. If only one agent has *yes*, the agent becomes the follower. When *type* is not *none* but both the agents have *no*, an agreement is not established.

hCond Denotes host requirements of each agent.

According to the matching rules described above, all the possible patterns of agreements are shown in Table 3.3.

3.4 Behavior of Each Agent

In addition to agreements and policies described in the previous section, abstract and general models have been defined in Event Calculus. Due to space limitation, intuitive descriptions are given for them.

– Common behavior of agents to comply with established agreements is defined so that agents always try to migrate accordingly when they become obliged to follow migration of a partner.

– Each agent has its own requirements on host resources, which activates (non-cooperative) mobility. Requirements on host resources may change along an agent's execution, depending on the current task. This characteristic is expressed in Event Calculus by introducing a fluent that denotes the current host requirements and events initiating/terminating it (events that denote change in host requirements).
– Design-time constraints and runtime modifications of policies are introduced in order to avoid establishing agreements inconsistent with existing agreements with other agents or the agent's own migration behavior. Intuitively, they prevent an agent from following one of its partners if the agent already has agreed to follow another partner or if the agent requires unique hosts rather than general host resources.

In Section 4, these models are discussed in detail through mapping to our implementation models.

4 Implementation

4.1 Freedia Framework

The proposed models are implemented on an existing agent framework, Freedia [7,8]. The Freedia framework was developed in the Smartive project, which aims for flexibly controlled distribution and provision of multimedia contents. For the purpose, the project has considered flexible and dynamic composition, deployment and provision of multimedia services by (possibly mobile) agents. The Freedia framework has the following features:

Service-Oriented. The main logic of an agent is specified in the same way as process descriptions for services such as BPEL [17], that is, interaction with partners are described without specifying their concrete binding information. Control- and data-flow among activities (action units) are described in a graph-oriented or a procedural way.

Policy-based. Given such a process description, an agent can run as an ordinary service-composing/providing/consuming agent with partner references statically given in advance. Flexible and dynamic (often complex) behaviors can be gradually introduced by incorporating policy descriptions provided for each aspect. For example, policies for customization of runtime discovery/selection strategies and rebinding upon events (e.g., user movement) have been provided [7,13]. A notion called *partner scope* is introduced, intuitively, a set of activities where interaction with a partner can occur.

Below policies to control non-cooperative and cooperative mobility of agents are described, respectively.

4.2 Migration Policy Descriptions

Migration Policy descriptions are given to an agent to specify the agent's own requirements on Places, or host resources. As effectiveness of migration depends

```
<migrate block="activities">
    <Target>target</Target>
</migrate>
```

Fig. 5. Entry in Migration Policy Descriptions

on surrounding environments, separated policy descriptions are given to control migration behavior. Figure 5 illustrates the structure of an entry in Migration Policy descriptions.

Execution Block. In Migration Policy Descriptions, migration behavior is associated with an *execution block* (the *block* attribute in Figure 5). An execution block is intuitively a set of activities that are executed in succession. An execution block is defined as a set of activities that make a connected graph, of control links, that starts from one activity. This activity is called the *start activity* of the block, and is always executed first in the block because a graph of control links is acyclic. A block can nest in another block recursively but not cross the boundary of another block.

Migration is conducted just before execution of the start activity of the block. A notable point here is that an execution block is associated with migration, not a point to insert a migration action. Constraints can be explicitly specified that an agent wants to stay on the target host, after migration, until it finishes some task that should be done on the host.

Migration Target. The *Target* element in Figure 5 indicates how to determine target of the migration, and includes either of the following descriptions.

Host Requirements. Specify requirements on hosts. The agent migrates to a Place that satisfies the specified requirements upon entering the execution scope, only if the current Place does not satisfies the requirements. The provided vocabulary currently includes CPU, memory, disk space, and provided services. Vocabularies for security requirements, such as trust value, are under discussion.

Static References. Specify an ordered list of references to specific Places. The agent tries to migrate to one of the specified Places, one by one in the order of the list.

Reference to Decision Mechanism. Specifies a reference to a Java class or another agent that implements complex decision mechanism by using query API to DFs and returns an ordered list of references to Places. Optimization methods can thus be incorporated, especially, it is possible to introduce global methods that consider requirements of multiple agents when an agent is referred to as implementation of decision mechanism. Regarding implementation of decision mechanism as a Java class, an abstract class is provided and API can be used to query DFs and to obtain requirements of all the follower agents obliged to follow the agent directly or indirectly (through chain of obligation). Regarding implementation of decision mechanism as

```
<InteractionStyle>
    <Nearby>strong/weak/none</Nearby>
    <CanFollow>yes/no</CanFollow>
</InteractionStyle>
```

Fig. 6. Policy Descriptions for Cooperative Mobility

another agent, abstract classes are provided to implement query behavior of the migrating agent and reply behavior of the decision agent.

4.3 Cooperative Mobility Policy

For each partner, policies for cooperative mobility are given, combined with policies for discovery, selection and negotiation proposed in the authors' previous study [13]. They include just descriptions of the *nearby* and *canfollow* parameters defined in Section 3.3. The default values are *none* for *Nearby* and *no* for *CanFollow*, that means, agents never migrate for partners by default.

4.4 Policy Constraints

Requirements of *type* (*strong*, *weak*, or *none*) should be determined according to effects of defined locality in the application domain. If locality defined physically or logically leads to availability of resources or services, locality is required for effective interaction (*type* should be *strong*). If locality leads to optional improvement of performance, it can be optional (*weak*). On the other hand, decision of which agent follows the other should be analyzed carefully, as involved obligation and prohibition can conflict with those given by other agreements or agents' own requirements.

An agent may have such conflicts may in situation where multiple agreements are effective or when migration by its own requirements is conducted while some agreement(s) are effective. Agent policies should thus be adjusted to avoid conflicts, as described below.

– In agreements between agents (Section 3.3), host requirements of follower agents are included so that the followed agents are aware of them. However, as the requirements are declared statically upon establishment of agreements, an agent should declare not to be the follower of a partner if it has change in host requirements during interaction with the partner. In the current implementation, this constraint is checked by alerting upon input of policies, if:
  - A partner scope includes a start activity of execution block,
  - The start activity is not always executed first in the partner scope,
  - And *CanFollow* for the partner is set as *yes*.
– When host requirements of agents are general, they can be satisfied together with locality requirements given by agreements, e.g., hosts that can provide specified amount of CPU and memory. However, some agents may have

specific requirements that cannot be satisfied together with locality requirements, e.g., direct indication of IP addresses, including "never move from the current host". If an agent have such specific requirements, it should declare not to be the follower. In the current implementation, this constraint is checked by alerting upon policy input, if:
  - An activity is included both in a partner scope and in an execution block,
  - Migration target for the block is given not by host requirements,
  - And *CanFollow* for the partner is set as *yes*.
- In general, it is impossible for an agent to be local with multiple agents. An agent should thus avoid being the follower in multiple agreements. In the current implementation, when an agent establishes an agreement where it follows another agent, policies with the other partners are overridden by setting *CanFollow* as *no* until the agreement is discarded.

With these constraints on policies, there is no occurrence of conflicts in permission and obligation managed for cooperative mobility. Especially, there is no situation where an agent try to migrate, according to its own Migration Policy or obligation given by established migration, but is prohibited to do so. Such consistency properties have been discussed and proved on the formal models specified in Event Calculus, which is not described due to the space limitation, though.

5 Discussion

In the proposed framework, agreements are introduced to handle cooperative mobility. Necessity of agreements has already been discussed in Section 2.2. Below decrease in development burden is discussed.

Cooperative migration is programmed by specifying requirements and constraints of each agent. Here an example scenario is described.

- The requester agent sends a message in order to request the counterpart to follow its migration.
- The responder agent waits for such a request, determines whether to accept or not, and replies to the request. If agreed, the responder agent activates *follow* commands.
- When the (followed) requester agent migrates, migration of the other is activated by the platform as implementation of the commands. However, a runtime error may occur due to migration failure or lack of hosts satisfying the follower's requirements. The two agents should catch such an error to handle it.

In the existing programming model, these behaviors are implemented by combination of primitive functions for messaging as well as *follow* commands. This study has extracted common behavior in such a flow and allowed developers only to describe customization parameters, eliminating the necessity for each developer to implement such a flow.

Below it is discussed how heavy loads the Freedia framework decreases in design and implementation of agreement-based cooperative mobility.

Strategies for Agreement Establishment. It is necessary to determine strategies for agreement establishment, or policies for cooperative mobility. This study has proposed and implemented constraints in such policies to avoid inconsistency in agreements so that application developers can avoid considering the consistency problem and concentrate on strategy selection itself. In this study, design using a formal model has required discussion on about 80 statements in Event Calculus. Such design is a heavy load when it is imposed on every application developers. Formal design of agreement-related behaviors is thus one of significant contributions of this study. On the other hand, once strategies of policy modification to avoid inconsistency are clarified, it is not so difficult to implement them. Their implementation has required about 50 lines of Java codes in Freedia, which is not so heavy a load even if imposed on each application developer.

Interaction for Agreement Establishment. It is necessary to implement interaction for agreement establishment through policy matching. Although the Freedia framework has implemented the behavior, the decreased load is not so heavy. The behavior corresponds to about 150 lines of Java codes in Freedia.

Agreement Enforcement. It is necessary to implement behavior to comply with established agreements, or behavior for cooperative mobility. It has required about 400 lines of Java codes in Freedia, which is quite a heavy burden if imposed on each application developer. In addition, the implementation codes appear in various parts in agent behavior, leading to very complex codes, due to necessity of state management in weak mobility [1]. Implementation of migration behavior in complying with agreements is thus one of significant contributions of this study.

Agreement Monitoring. If enforcement of agreements is not conducted by the underlying platform, each agent needs to monitor whether other agents is acting in complying with established agreements. In the case of cooperative mobility in this study, a followed agent checks its partner actually follows it and does not get away. A following agent, on the other hand, checks it is not obliged to move to a host that does not satisfy the agent's requirements. Although these monitoring actions are not actually in the Freedia framework, implementation for evaluation required about 200 lines of Java codes. Although implementation task is not so hard, it changes the background greatly in a sense application developers need to consider possibility of agreement violation at any time.

As discussed above, the approach in this study decreases loads in design and implementation of supported patterns for cooperative mobility. However, this approach limits flexibility in behavior, supported behavior should continue to be discussed.

[1] It is necessary to preserve/restore states on instance variables in Java mobile agents as program counters cannot be preserved upon migration.

6 Related Work

6.1 Cooperative Mobility

SyMPA [3] and MobileSpaces [4] are agent platforms that support the hierarchical model of cooperative mobility (Section 2.1). They provide API with which an agent can enter into or exit out of another agent. Although the metaphor is intuitive, that command directly supports only situations where cooperative migration is initiated by the following (contained) agent. However, an agent often asks another to follow, as discussed in Section 2.2. This study has considered agreements between agents, which are essentially established by reflecting each of their policies. MobileSpaces also introduces strong controls on a contained agent by the containing agent. However, as no coordination mechanism is provided, it is difficult to make use of such controls when agents are provided by different parties. This study has considered agreements on such strong controls so that agents can expect controls they impose or receive.

The loosed model of cooperative mobility is introduced in [5]. It only provides commands to "follow" migration of another agent. There are the same problems as in SyMPA and MobileSpaces.

Many studies have considered composition of services by combining multiple services, especially in the recent activity on service-oriented computing [18]. However, composition of mobile agents by combining multiple agents has not investigated so much.

Multimedia application, including exchange of large amount of data, is originally one of attractive application areas for mobile agents. Naturally, cooperative mobility has been used for composition of multiple multimedia contents. MobileSpaces has also been adopted to multimedia application [6]. Agents encapsulate multimedia contents and form a hierarchy, which corresponds to the hierarchy of the multimedia contents, e.g. images included and managed by a word processor document. The agents provide to users, customized functions to operate on multimedia objects (play, edit, etc.). In addition, required services can be added at runtime as contained agents, e.g. streaming functions.

6.2 Agreements

The notion of agreements/contracts has been investigated in Service-Oriented Computing and Multi-Agent Systems communities. Most of their studies has considered permissions and obligations, often together with their state transition by modeling them in Event Calculus[19,20,21]. In most cases in Service-Oriented Computing, Service Level Agreements (SLA) are just declared by service providers, and it has been discussed how service implementation enforce the agreements [19,20]. On the other hand, there is a study in multi-agent system that handles agreements as knowledge that can be used for reasoning to achieve agent's goals [21]. There flexible vocabularies are handled such as agreements that will be activated in a certain future time point and clarification of the subject that is responsible to change the state of predicates (fluents). Such flexibility remains as future work in our study for cooperative mobility.

In generally, it is necessary for each application developer to define targets of permission and obligation, events/actions that cause changes in permission and obligation, and so on, according to the target domain. This study has focused on the domain of cooperative mobility, and provided vocabularies, axioms, and theorems (though omitted), eliminating the necessity for each application developer to do that. Another unique point is that this study has covered the whole lifecycle of agreements: from establishment to enforcement and discard, considering both sides of the participants in agreements.

7 Summary

This study has proposed a model for agreement-based cooperative mobility in mobile agents and incorporated it into the Freedia framework for development of service-oriented agents that discover and cooperate with each other at runtime. Agent behavior defined in the proposed model involves agreement establishment and enforcement for cooperative mobility, and can be customized only by specifying requirements and constraints of agents, eliminating necessity to write down the whole behavior to handle agreements. The model is defined in Event Calculus, and is proved to have no occurrence of defined inconsistency. The Freedia framework is being adopted to various scenarios and refined continuously.

References

1. Milojicic, D.: Mobile agent applications. IEEE Concurrency 7(3), 7–13 (1999)
2. Chess, D., Harrison, C., Kershenbaum, A.: Mobile agents: Are they a good idea? Technical Report RC 19887, IBM TJ Watson Research Center (1994)
3. Suna, A., Fallah-Seghrouchni, A.E.: A mobile agents platform; architecture, mobility and security elements. In: Bordini, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F. (eds.) Programming Multi-Agent Systems. LNCS (LNAI), vol. 3346, Springer, Heidelberg (2005)
4. Satoh, I.: Mobilespaces: A framework for building adaptive distributed applications using a hierarchical mobile agent system. In: ICDCS 2000. The 20th International Conference on Distributed Computing Systems, pp. 161–168 (April 2000)
5. Satoh, I.: Organization and mobility in mobile agent computing. In: Bordini, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F. (eds.) Programming Multi-Agent Systems. LNCS (LNAI), vol. 3862, pp. 187–205. Springer, Heidelberg (2006)
6. Satoh, I.: Mobile agent-based compound documents. In: ACM Symposium on Document Engineering 2001, pp. 76–84. ACM Press, New York (2001)
7. Ishikawa, F., Yoshioka, N., Honiden, S.: Smartive: Agreement-based mobile composite agents for multimedia services. In: IAWTIC 2006. International Conference on Intelligent Agents, Web Technologies and Internet Commerce (November 2006)
8. Smartive project: Smartive.jp. (February 2007 (Last Access)), http://smartive.jp/eng/index.htm
9. Satoh, I.: Bio-inspired deployment of distributed applications. In: Barley, M.W., Kasabov, N. (eds.) PRIMA 2004. LNCS (LNAI), vol. 3371, pp. 243–258. Springer, Heidelberg (2005)
10. Singh, M.P., Huhns, M.N.: Service-Oriented Computing: Semantics, Processes, Agents. John Wiley and Sons, England (2005)

11. Forum, G.G.: Web services agreement specification (ws-agreement) (September 2004), www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graap-agreement.pdf
12. Jin, L.j., Machiraju, V., Sahai, A.: Analysis on service level agreement of web services. Technical Report HPL-2002-180, HP Labs (July 2002)
13. Ishikawa, F., Yoshioka, N., Honiden, S.: Policy-based runtime partner management in process-based services. In: ICWS 2007. 2007 IEEE International Conference on Web Services (2007)
14. Bellavista, P., Corradi, A., Stefanelli, C.: Mobile agent middleware for mobile computing. Computer 34(3), 73–81 (2001)
15. Shanahan, M.: The event calculus explained. Artificial Intelligence Today, 409–430 (1999)
16. Bandara, A.K., Lupu, E.C., Russo, A.: Using event calculus to formalise policy specification and analysis. In: 4th IEEE international workshop on policies for distributed systems and networks, pp. 26–39 (2003)
17. Thatte, S., et al.: Business process execution language for web services, version 1.1 (May 2003), http://www.ibm.com/developerworks/library/specification/ws-bpel/
18. Haas, H.: Web services (June 2004) (Access: May 2005), http://www.w3.org/2002/ws/
19. Farrell, A.D.H., Sergot, M.J., Salle, M., Bartolini, C.: Performance monitoring of service-level agreements for utility computing using the event calculus. Technical report, HP Labs (November 2004)
20. Paschke, A., Dietrich, J., Kuhla, K.: A logic based sla management framework. In: ISWC Semantic Web and Policy Workshop (November 2005)
21. Knottenbelt, J., Clark, K.: Contract-related agents. In: Computational Logic in Multi-Agent Systems, 6th International Workshop (CLIMA VI), pp. 226–242 (June 2005)

A Example of Formal Model in Event Calculus

Here one example of the formal model in Event Calculus is described to give an intuition. Figure 7 shows EC expressions of agreements for cooperative mobility (corresponding to Figure 4 in Section 3.3). Actions to establish and discard an agreement are defined as $agree_a$ and $discard_a$, respectively. A state where an agreement is effective is defined as fluent $agreed_a$ $(aINIT1, aFIN1)$. Arguments in these actions and fluents are the same as those in Section 3.3, as follows.

$a1$, $a2$ Denote agents that participate in the agreement. When cooperative migration is conducted, $a1$ (the first argument) becomes the follower.
type Denotes type of cooperative migration and is any of *strong*, *weak*, or *none*.
$hCond1$ Denotes host requirements of the follower, $a1$.
$tout_a$ Denotes time span within which $a1$ has to migrate following migration of $a2$ (when *type* is *strong* or *weak*).

In Figure 7, an event *followEvent(*$a1$*,* $h2$*)* is introduced to denote occurrence of any event leading to obligation on $a1$ to migrate to a host nearby $h2$ (aFE). The *followEvent* events happen in the following situations, when *type* is not *none*.

$$
\begin{array}{rl}
(aINIT1) & \text{Initiates}(\text{agree}_a(a1,\, a2,\, type,\, hCond1,\, tout_a), \\
& \qquad \text{agreed}_a(a1,\, a2,\, type,\, hCond1,\, tout_a),\, \tau) \\
(aINIT2) & \text{Happens}(\text{agree}_a(a1,\, a2,\, type,\, hCond1,\, tout_a),\, \tau) \wedge type \neq none \\
& \quad \wedge\ \text{HoldsAt}(\text{stayAt}(a1,\, h1),\, \tau) \wedge \text{HoldsAt}(\text{stayAt}(a2,\, h2),\, \tau) \\
& \quad \wedge\ \neg\ \text{HoldsAt}(\text{local}(h1,\, h2),\, \tau) \\
& \Rightarrow \text{Happpens}(\text{followEvent}_i(a1,\, h2),\, \tau + 1) \\
(aINIT3) & \text{Happens}(\text{agree}_a(a1,\, a2,\, type,\, hCond1,\, tout_a),\, \tau) \wedge type \neq none \\
& \quad \wedge\ \text{HoldsAt}(\text{stayAt}(a1,\, h1),\, \tau) \wedge \text{HoldsAt}(\text{stayAt}(a2,\, h2),\, \tau) \\
& \quad \wedge\ \text{HoldsAt}(\text{local}(h1,\, h2),\, \tau) \\
& \Rightarrow \text{Happpens}(\text{constrain}(a1,\, h2),\, \tau) \\
(aFE) & \text{Happens}(\text{followEvent}_i(a1,\, h2),\, \tau) \\
& \quad \vee\ \text{Happens}(\text{followEvent1}(a1,\, h2),\, \tau) \\
& \quad \vee\ \text{Happens}(\text{followEvent2}(a1,\, h2),\, \tau) \\
& \Leftrightarrow \text{Happens}(\text{followEvent}(a1,\, h2),\, \tau) \\
(aOBL) & \text{Initiates}(\text{followEvent}(a1,\, h2), \\
& \qquad \text{obliged}(a1,\, a1,\, \text{bind}_h(hCond1\&localTo(h2))),\, \tau) \\
(aPER1) & \text{Happens}(\text{followEvent}(a1,\, h2),\, \tau) \Rightarrow \text{Happens}(\text{constrain}(a1,\, h2),\, \tau) \\
(aPER2) & \text{HoldsAt}(\text{local}(h1,\, h2),\, \tau) \\
& \Rightarrow \text{Initiates}(\text{constrain}(a1,\, h2), \\
& \qquad \text{permitted}(a1,\, h1,\, \text{requestAgree}_h(hCond_r)),\, \tau) \\
(aPER3) & \neg\ \text{HoldsAt}(\text{local}(h1,\, h2),\, \tau) \\
& \Rightarrow \text{Terminates}(\text{constrain}(a1,\, h2), \\
& \qquad \text{permitted}(a1,\, h1,\, \text{requestAgree}_h(hCond_r)),\, \tau) \\
(aFIN1) & \text{Terminates}(\text{discard}_a(a1,\, a2,\, type,\, hCond1,\, tout_a), \\
& \qquad \text{agreed}_a(a1,\, a2,\, type,\, hCond1,\, tout_a),\, \tau) \\
(aFIN2) & \text{Initiates}(\text{discard}_a(a1,\, a2,\, type,\, hCond1,\, tout_a), \\
& \qquad \text{permitted}(a1,\, h1,\, \text{requestAgree}_h(hCond_r)),\, \tau) \\
(aFIN3) & \text{Terminates}(\text{discard}_a(a1,\, a2,\, type,\, hCond1,\, tout_a), \\
& \qquad \text{obliged}(a1,\, a1,\, \text{bind}_h(hCond)),\, \tau)
\end{array}
$$

$$
\begin{array}{rl}
\multicolumn{2}{l}{\text{HoldsAt}(\text{agreed}_a(a1,\, a2,\, type,\, hCond1,\, tout_a),\, \tau) \wedge type \neq none} \\
\multicolumn{2}{l}{\Rightarrow} \\
(aFE1) & \text{Happens}(\text{migrate}(a2,\, h2_s,\, h2_d),\, \tau) \\
& \quad \wedge\ \text{HoldsAt}(\text{stayAt}(a1,\, h1_s),\, \tau) \\
& \quad \wedge\ \neg\ \text{HoldsAt}(\text{local}(h1_s,\, h2_d),\, \tau) \\
& \Rightarrow \text{Happpens}(\text{followEvent1}(a1,\, h2_d),\, \tau) \\
(aFE2) & \text{Happens}(\text{forcedOut}(a1,\, h1_s),\, \tau) \wedge \text{HoldsAt}(\text{stayAt}(a2,\, h2),\, \tau) \\
& \quad \wedge\ \neg\ \text{Happens}(\text{migrate}(a2,\, h2_s,\, h2_d),\, \tau) \\
& \Rightarrow \text{Happpens}(\text{followEvent2}(a1,\, h2),\, \tau) \\
(aTIME) & type = strong \wedge \text{Happens}(\text{followEvent}(a1,\, h2_d),\, \tau) \\
& \quad \wedge\ \text{HoldsAt}(\text{stay}(a1,\, h1),\, \tau + tout_a) \\
& \quad \wedge\ \neg\ \text{HoldsAt}(\text{local}(h1,\, h2_d),\, \tau + tout_a) \\
& \Rightarrow \text{Happens}(\text{discard}_a(a1,\, a2,\, type,\, hCond1,\, tout_a),\, \tau + tout_a)
\end{array}
$$

Fig. 7. Formal Model of Agreement for Cooperative Migration

- When an agreement is established and $a1$ and $a2$ is not nearby ($aINIT2$).
- When migration of $a2$ occurs and locality becomes lost ($aFE1$).
- When $a1$ has to move to another host due to forcedOut events in the agreement with the current host ($aFE2$).

The event *followEvent(*$a1$*,* $h2$*)* leads to obligation of $a1$ to invoke *bind*$_h$ so that $a1$ discovers a host that satisfies its own requirements $hCond1$ as well as locality requirements($aOBL$). Invocation of *bind*$_h$ leads to establishment of an agreement with a host and then obligation to enter the host (defined in the description of host agreements in Event Calculus). If such following migration is not conducted within *tout*$_a$, the agreement is discarded when *type* is *strong* ($aTIME$).

In addition, the event *constrain(*$a1$*,* $h2$*)* occurs together with *followEvent(*$a1$*,* $h2$*)*. The event *constrain(*$a1$*,* $h2$*)* also happens when an agreement is established if *type* is not *none* and the agents already stay nearby. The event *constrain(*$a1$*,* $h2$*)* adjusts permission of $a1$ so that $a1$ can migrate only to hosts nearby $h2$ ($aINIT3, aPER1$, $aPER2, aPER3$).

When an agreement is discarded, given obligation and prohibition are terminated ($aFIN2, aFIN3$).

An Adaptive Coupling-Based Algorithm for Internal Clock Synchronization of Large Scale Dynamic Systems*

Roberto Baldoni[1], Angelo Corsaro[2], Leonardo Querzoni[1], Sirio Scipioni[1], and Sara Tucci-Piergiovanni[1]

[1] Dipartimento di Informatica e Sistemistica "A. Ruberti"
Sapienza - Università di Roma
Rome, Italy
{baldoni,querzoni,scipioni,tucci}@dis.uniroma1.it

[2] PrismTech
4, Rue Angiboust, 91460
Marcoussis, France
angelo.corsaro@prismtech.com

Abstract. This paper proposes an internal clock synchronization algorithm which combines the gossip-based paradigm with a nature-inspired approach coming from the *coupled oscillators* phenomenon. The proposed solution allows a very large number of clocks to self-synchronize without any central control, despite node departure and arrival. This addresses the needs of an emergent class of large-scale peer-to-peer applications that have to operate without any assumptions on the underlying infrastructure. Empirical evaluation shows extremely good convergence and stability under different network settings.

1 Introduction

Clock synchronization is a fundamental building block for many distributed applications. As such, the topic has been widely studied for many years, and several algorithms exist which address different scales, ranging from local area networks (LAN), to wide area networks (WAN). For instance, the Network Time Protocol (NTP) [23,24], has emerged as a standard de facto for external clock synchronization in both LAN and WAN settings. The work presented in this paper is motivated by an emergent class of applications and services, operating in very challenging settings, for which the problem synchronizing clocks is far from being solved. These applications are required to (1) operate without any assumption on deployed functionalities, pre-existing infrastructure, or centralized control, while (2) being able to tolerate network dynamism, due to crashes or to node joining or leaving the system, and (3) scaling from few hundred to tens of thousands of nodes. For instance, publish/subscribe middleware, such as the data distribution service [1] requires synchronized clocks, however in several relevant scenarios,

* The work described in this paper was partially supported by CINI-Finmeccanica and the EU Project Resist.

due to security issues, or limited assumptions on the infrastructure, it cannot assume that members of the system, either have access to an NTP server, or are equipped with an NTP daemon.

A promising approach to tackle this kind of problems is to embrace a fully decentralized paradigm in which peers implement all the required functionalities, by running so called $gossip - based$ algorithms. In this approach, due to the large scale and geography of the system, each peer is provided with a neighborhood representing the part of the system it can directly interact with. The algorithm running at each peer computes local results by collecting information from this neighborhood. These results are computed periodically leading the system to gradually compute the expected global result. In this paper, in order to obtain clock synchronization, we combine this gossip-based paradigm with a nature-inspired approach coming from the *coupled oscillators* phenomenon. This phenomenon shows enormous systems of oscillators spontaneously locking to a common phase, despite the inevitable differences in the natural frequencies of the individual oscillators. Examples from biology include network pacemaker cells in the heart, congregations of synchronously flashing fireflies and crickets that chirp in unison. A description of the phenomenon was pioneered by Winfree [2]. He mathematically modeled a population of interacting oscillators and discovered that assuming nearly identical individual frequencies and a certain strength of the coupling (which is a measure of the sensitivity each oscillator has to interactions with others), a dramatic transition to a globally entrained state, in which oscillators freeze into synchrony, occurs. A valuable contribution has been subsequently introduced by Kuramoto [3] who simplified the Winfree model by considering the coupling strength constant for all oscillators and depending only on their phase difference. Both Winfree's and Kuramoto's work was done assuming that each oscillator is coupled directly and equally to all others, which means assuming a fully connected oscillators network. However a considerable amount of work has been done also on so called "non-standard topologies". Satoh in [5] performed numerical experiments comparing the capabilities of networks of oscillators arranged in two-dimensional lattices and random graphs. Results showed that the system becomes globally synchronous much more effectively in the random case. In fact, Matthews et al. in [6] note that the coupling strength required to globally synchronize oscillators in a random network is the same as the one required in the fully interconnected case.

In this paper we adapt the Kuramoto model to let a very large number of computer clocks synchronize over a random graph. The first issue we tackle is how to artificially reproduce the physical phenomenon in a network of computer clocks in which every clock can be influenced by other clocks only by exchanging messages reporting local values. In our approach, each clock (process) explicitly asks clock values from neighboring processes in order to calculate their difference in phase. Then, following our Kuramoto-like model, these differences in phase are combined and multiplied by a so-called *coupling factor*, expressing the coupling strength, in order to adjust the local clock.

As the coupling factor has a key role in regulating the dynamics of coupling, we study thoroughly its impact on the performance of the proposed solution. First, we consider a time-invariant coupling factor identical for all oscillators. Different constant coupling factors are then evaluated to investigate their effect on system perturbations specific to target settings (basically deployed on a wired computer network): (1) *errors on the phase difference estimates due to network delays* and (2) *neighborhoods possibly changing over time*. As a general result, low coupling factors lead to better synchronization regardless of system perturbations–all clocks lock to a value such that their differences are negligible. On the other hand, higher coupling factors lead to a faster locking at the cost of more dispersed values. This phenomenon depends on the fact that a higher coupling factor augments the sensitivity a clock has with respect to other clocks but it also increases the influence of system perturbations.

Another fundamental aspect this approach revealed is its unprecedented scalability: the time to converge to a common value remains the same considering both a few dozen nodes and thousands nodes, with even a small reduction in the latter case.

Even though these observations are really encouraging per se, we further improve the system behaviour by using an *adaptive* coupling factor, with the objective of reducing the impact of system perturbations while still keeping the time to converge small. This new approach has been revealed really successful, both in the case of errors phase estimates due to network delays and in the case of changing neighbors. The idea is simple: the local coupling factor reflects the age of a node (expressed by the number of adjustments already performed); a young node will have a high coupling factor to absorb soon values from other nodes, while an old node will have a small coupling factor to limit its sensitivity to system perturbations. The rationale behind this mechanism comes from the observation that an old node is more aligned to the values of other clocks than a new one. With this adaptive coupling factor, a young node, supposed to have a value generally far from other clock values, will rapidly align its value to others since the system perturbations have a small impact when the relative clock differences are still huge. Then, when nodes reach good values, i.e. their relative differences are small, a lower coupling factor lets to maintain these differences small despite system perturbations. This strategy reveals to be particularly useful in case of a dynamic system. Considering a network which starts and locks to some clock value, the perturbation caused by a massive entrance of new nodes (generally not synchronized with the ones which already reached a synchronization inside the network) could be dramatically reduced when compared to a constant coupling factor. In other words, the adoption of adaptive coupling leads the system to maintain its stability, a property strongly needed in face of network dynamism. The rest of the paper is organized as follows: Section 2 presents the clock coupling model along with the algorithm. The experimental evaluation is presented in Section 3. Section 4 discusses related works, while Section 5 concludes the paper.

2 Clock Coupling Model

Every computer is equipped with a hardware clock consisting of an oscillator and a counting register that is incremented at every tick of the oscillator. Depending on the quality of the oscillator, and the operating environment, its frequency may drift. Manufacturers typically provide a characterization for ρ – the maximum absolute value for oscillator drift. Ignoring, for the time being, the resolution due to limited pulsing frequency of the oscillator, the hardware clock implemented by the oscillator can be described by Equation 1:

$$C(t) = ft + C_0; \tag{1}$$

where: $(1 - \rho) \leq f \leq (1 + \rho)$ This clock model is assumed by all clock synchronization algorithms described in literature, thus we will also adopt it, and we won't go in further details in characterizing its properties. In the remainder of this Section we will describe mathematical principles governing the *coupling* of different clocks in order to synchronize them.

2.1 Time Continuous Clock Coupling

Let us consider a system composed by N distinct clocks, with each clock C_i being characterized by a frequency $f_i \in [1 - \rho, 1 + \rho]$, and characterized by equation (1). Clocks are initially non synchronized, meaning that they might show different time readings at the same real-time.

Thanks to a continuous coupling of these clocks over time, they will lock to a so-called stable point: each clock will show the same value, without changing the value once reached.

Even though our coupling resembles the model proposed by [3,4], it is worth noting that Kuramoto modeled a non-linear oscillator coupling which is not directly applicable to our problem. In fact, the non-linear oscillator used by Kuramoto to model the emergence of fireflies flashing synchrony, models intentionally a phenomenon which is characterized by several stable points (which arise due to the sinusoidal coupling), *i.e.*, the system does not converge to a unique point, but it can partition in subsystems each with a different stable point. On the other hand, for synchronizing clocks in a distributed system it is highly desirable that a single point of synchronization exists. This leads to consider a *linear* coupling equation of the form:

$$\dot{C}_i(t) = f_i + \frac{\phi_i}{N} \sum_{j=1}^{N} (C_j(t) - C_i(t)), \quad i = 1..N \tag{2}$$

The intuition behind Equation 2 is that a clock has to speed up if its neighboring clocks are going faster, while it has to slowdown when they are going slower. The coupling constant ϕ_i provides a measure of how much the current clock rate should be influenced by others. It can be shown analytically that Equation 2 has a single stable fixed point, and thus converges, in the case in which all the clocks are connected to each other. An empirical evaluation of the convergence of Equation 2, under different topologies, can be found in [28].

2.2 Time Discrete Coupling with Imperfect Estimates

The coupling model described in Equation 2 is not directly applicable to distributed systems as (1) it is based on differential equations, and thus continuous time, and (2) it assumes that the underlying topology is a fully connected graph. As shown below, Equation 2 can be made applicable to distributed systems by (1) considering its discrete counterpart, and (2) explicitly introducing a dependency on the underlying communication graph.

$$\begin{aligned} C_i(n+1) = {} & C_i(n) + f_i \varDelta T + \\ & + \frac{K_i}{N_i} \sum_{j=1}^{N_i} [(C_j(n) - C_i(n)) * edge(i,j)], \\ & i = 1..N \end{aligned} \tag{3}$$

Where $K_i = \phi_i f \varDelta T$, N_i is the number of neighbors for clock C_i, and $edge(i,j)$ is 1 when the underlying communication graph has an edge (i,j), 0 otherwise. When applying Equation 3 in real distributed systems, the clock difference $(C_j(n) - C_i(n))$ will be estimated with an error ϵ which depends on the mechanism used to perform the estimation. In this paper we assume that the difference between neighboring clocks are estimated as NTP does [23,24] (see Figure 1). Under this assumption, the real offset between C_i and C_j is such that the error is $(\delta_1 - \delta_2)/2$. Note that, if the two delays are equal (channel symmetry) the error is zero. Moreover, it has been shown that the maximum error is bounded by $\pm(\delta_1 + \delta_2)/2 \approx \pm RTT/2$, where RTT is the round trip time. Thus, we can now rewrite Equation 3 by considering the error which affects the $(C_j(n) - C_i(n))$ estimation:

$$\begin{aligned} C_i(n+1) = {} & C_i(n) + f_i \varDelta T + \\ & + \frac{K_i}{N_i} \sum_{j=1}^{N_i} [(C_j(n) - C_i(n)) * edge(i,j)] + \\ & + \frac{K_i}{N_i} \sum_{j=1}^{N_i} [(\frac{\delta_{i,j} - \delta_{j,i}}{2}) * edge(i,j)], \quad i = 1..N \end{aligned} \tag{4}$$

Equation 4 shows how $K_i < 1$ helps reducing the sensitivity of the coupling model on the estimation error. On the other hand, K_i close to 1 provides faster convergence time, as we will see later in the paper, while values greater than 1 make the system diverge. As a final remark, it is worth pointing out that K_i could be made time dependent, and as we will see in the reminder of the paper, this could be very useful for making the solution more robust and performant.

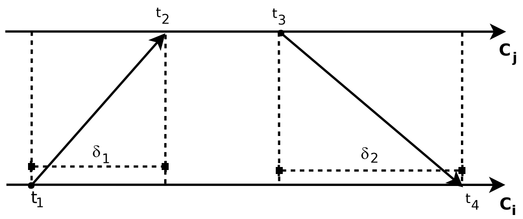


Fig. 1. NTP offset estimation

Finally, if we consider the worst case bound on estimate error, it is worth pointing out that slow channels (high RTT) may introduce more noise than fast channels (low RTT), however, it is important to keep in mind that the source of error is not the RTT per se, but the asymmetry, *i.e.*, the difference between δ_1 and δ_2.

2.3 The Clock Coupling Algorithm

In order to translate Equation 4 into an algorithm, we should note that the value C_i is computed periodically, every ΔT. As a result, the coupling-based clock synchronization algorithm proceeds in synchronization rounds, performing at each round the following steps:

1. Evaluate the difference with every neighboring clock, using a request-reply pattern of messages.
2. Sum the differences and multiply by $\frac{K_i(n)}{N_i}$.
3. Update the value of C_i, and the value of K_i, in the case in which it is time dependent.

In the following we provide the results of an extensive experimental study which highlights the behaviour, convergence time, synchronization error, and stability, of the proposed algorithm.

3 Empirical Evaluation

The aim of this section is to show the behaviour of the proposed coupling algorithm when the different clocks are connected by a communication graph obtained through a peer-sampling service [26]. The mechanism evaluation is performed defining a set of metrics and then studying, through simulations[1], their evolution in a set of scenarios each defined with the aim of isolating some specific aspects. In the reminder of this Section we will first describe the details of the simulation settings used for our evaluation, and then the final performance results.

3.1 Simulation Settings

The proposed algorithm is evaluated against the metrics and the scenarios described below and summarized in Table 3.1.

Table 1. Summary of evaluated scenarios, with metrics measured against which parameters

	Static Symmetric	Static Asymmetric	Dynamic Symmetric
Convergence Time	K, N	–	–
Synchronization Error	–	K, Asymmetry	K, Stable Core%
Stability	-	–	K, Injected nodes%

[1] Tests were run on Peersim, a simulation software specifically designed to reproduce large-scale scenarios. The simulator code for the coupling mechanism is available for download at the following address: `http://www.dis.uniroma1.it/~midlab/clocksync_sim.zip`

Evaluation Metrics. The metrics used to evaluate the proposed algorithm are its convergence time, synchronization error, and stability. A precise definition, for each metric, is provided below.

Convergence Time. The convergence time metric is defined as the number of synchronization rounds (SR) taken to converge to a desired standard deviation of clocks. More specifically, in our tests we will measure the number of synchronization rounds taken to reach a standard deviation equal to $10\mu sec$. It should be noticed that we consider SR as the only convergence time metrics as, once the duration ΔT of a synchronization round has been fixed, the time to reach a predefined clock dispersion only depends on the number of synchronizations.

Synchronization Error. The synchronization error (SE) is defined as the minimum standard deviation the algorithm can achieve. This metric will not be considered in the static scenario with symmetric channels, as for symmetric channels the estimation error on clock differences is zero, and thus the synchronization error (see Equation 4). On the other hand this metric will be evaluated in both scenarios which include system perturbations: static with asymmetric channels and the dynamic scenario.

Stability. Once all clocks converge to some value, the stability metric measures how much these clocks are sensitive to the injection of new nodes. A perfectly stable algorithm should not allow these clocks to significantly change the convergency value already reached.

Simulation Scenarios. Below are described the scenarios in which the metrics, defined above, will be evaluated. These scenarios were defined so to isolate aspects relevant to the metrics under exam.

Static with Symmetric Channels. This scenario corresponds to a system in which the network is static (no nodes are added/removed during the simulation) and (1) the network delay is bounded but unknown, (2) communication channels are symmetric $\delta_1 = \delta_2$ and thus no estimation error occurs, (3) processes execute the algorithm periodically every ΔT time units, but not in a lock step mode. The round-trip time (RTT) is modeled by means of a Gaussian distribution whose mean and standard deviation have been derived, as described in [27] by fitting several round-trip data set measured over the Internet. To be more precise, in this scenario we consider two different kind of channels, slow and fast. Slow channels are characterized by an average round trip delay of $180msec$, while fast channels are characterized by an average round trip delay of $30msec$. When generating a communication graph links between nodes are randomly chosen to be of one kind or another. ΔT is the third quartile of the slow channels RTT Gaussian distribution. This model is worth considering as it closely matches the model underlying Equation 3.

Static with Asymmetric Channels. This simulation scenario adds to the previous one communication channel asymmetry. Channel asymmetry defines how the round-trip time is distributed between the two ways of a channel (*i.e.* given a channel connecting A to B, which is the ratio between the transfer delay of a message from A to B and the delay back from B to A). The asymmetry is modeled by means of a Gaussian distribution with mean 0.5 (*i.e.*, symmetric channel). The parameters of this distribution are used in order to explore the sensitivity of the algorithm to channel asymmetry, and thus to estimate clock difference errors.

Dynamic with Symmetric Channels. The last scenario considered in our tests takes into account network dynamics, and thus considers the evolution of a network under the continual addition/removal of nodes. In order to characterize only the dependency of the proposed algorithm under dynamics, we ignore the estimation errors on clock differences, thus assuming symmetric channels.

Simulation Parameters. Tests have been conducted varying both the number of clocks N, and the coupling factor K. In order to test our approach under different system scales, ranging from very small to very large, we will be considering values of N in the set of $\{8, 16, \ldots, 64K\}$. To show the dependency with respect to a constant coupling factor K we will consider values in the set $\{0.1, 0.2, \ldots, 1\}$. Tests aimed at evaluating the adaptive coupling factor will consider the following local coupling factor K_i behaviour: initially K_i assumes the 1 value, then it undergoes an exponential decay up to the point it reaches the 0.1 value. Specific tests aimed at evaluating the dependency of the synchronization error on channel asymmetry have been conducted varying the amount of asymmetry, either using a fixed value in the set $\{0.1, \ldots, 0.5\}$, or varying the variance of the Gaussian distribution used to model it within the set $\{10^{-3}, \ldots, 10^{-11}, 0\}$. Tests for the dynamic symmetric scenario have been conducted varying either the size of the stable core, *i.e.* the amount of nodes that remain in the system from the beginning to the end of the test, or the amount of replaced nodes for a single time unit. In all our tests we assume that the initial value assumed by a clock, referred as X_0, is a uniform random number in the interval $[0, 60]$ sec.

3.2 Static Scenario with Symmetric Channels Results

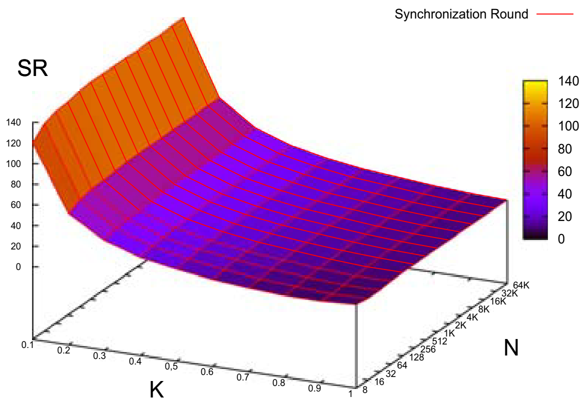
Assuming as a deployment scenario a static network with symmetric communication channels, we show how the convergence time, of the proposed algorithm, depends on the scale N, and on the coupling factor K–the case of K adaptive is also considered.

Convergence Time. Figure 2(a), shows how the synchronization rounds SR depend on the size of the system N, and on the coupling factor K. As it can be seen from the plot, given a value of N, there is negative exponential dependence K and SR. This dependence, can roughly be approximated an inverse dependence between K and SR, as (see Figure 2(a)) doubling K almost halves SR. On the other hand, if we fix the value of K we can see how SR grows slightly with N when $K \geq 0.5$, while it remains constant, or slightly diminishes with N when $K > 0.5$.

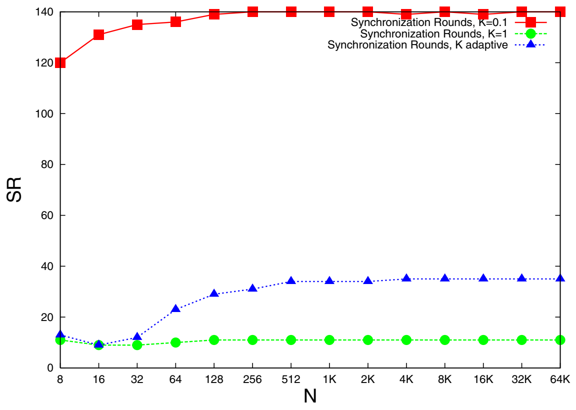
Figure 2(b) compares the effect of K adaptive on the convergence time. To this end, it shows the dependence of SR on the network size for $K = 1$, $K = 0.1$ and K adaptive. From this plot it is easy to see how K adaptive provides a performance improvement with respect to the convergence time that is close to that of $K = 1$, while, as shown later in the paper, retaining error mitigation properties similar to that of $K = 0.1$ (see Equation 4).

Synchronization Error. As the communication channels are symmetric, and thus the neighboring clock estimate is perfect, the synchronization error will tend to zero as a negative exponential (this comes from Equation 2).

Scalability. Most distributed algorithms tend to degrade their performance as the scale of the systems grows, when this happens, the scale of a system can practically exclude certain algorithmic solutions. Thus, it is extremely important to characterize what happens to a distributed algorithm as the number of nodes involved in the computation grows. To this end Figure 2(a) and Figure 2(b) show how the synchronization rounds SR change over a very wide set of network sizes. Contrarily to many existing clock synchronization solutions, the proposed algorithm scales extremely well, and in some cases, namely for $K > 0.5$, its performance slightly improve with N. This is a very important property which makes this solution appealing for applications that need to scale to a very large number of nodes. While the behaviour might seem counter-intuitive at first, it has a simple explanation. As the scale of the system grows, so does the average



(a) Static K



(b) Adaptive K

Fig. 2. Convergence dependency on N and K

rank of a node, *i.e.*, number of links, with the result that its random sample of neighbors has an increasingly higher probability of being closer to the system clock average, thus reducing the number of rounds taken to converge.

3.3 Static Scenario with Asymmetric Channels Results

Assuming as a deployment scenario a static network with asymmetric communication channels, we investigate how the asymmetry impacts on the synchronization error within which clocks synchronize. For this scenario we won't show results for the convergence time as these are analogous to what is described in the previous Section.

Synchronization Error. Figure 3 reports results obtained using a fixed value of asymmetry for all communication channels. The system size N is fixed to $64K$ for this plot. As the plot shows there is (1) a linear dependency between the channel asymmetry and the synchronization error, and (2) the value of K, as predicted by the Equation 4 behaves as scaling factor on the synchronization error. The results obtained with the use of K adaptive, are not shown as completely overlap with those found for $K = 0.1$. This should come at no surprise as the K after a transitory assumes definitively the value 0.1–the only relevant difference is that, as shown in Figure2(b), the use of K adaptive leads to shorter convergence times. The specific scenario used for the previous plot is far from being realistic, as it assumes a fixed value for asymmetry. Thus, to better model realistic channel asymmetry, we used a Gaussian distribution with mean 0.5 and studied how systems with variable sizes behave with respect to synchronization error, varying the variance of the distribution. Results for slow and fast channels are reported respectively in Figures 4(a) and 4(b), where the clock standard deviation (expressed in seconds) is reported. As the graphs show, the more channels are "symmetric", *i.e.*, the more the asymmetry variance is low, the lower is the synchronization error with a clear exponential dependency. It is interesting to point out that the error difference between slow and fast channels quickly becomes negligible as soon as we consider fairly symmetric channels. These plots therefore confirm that the impact of RTT on synchronization error is not straight, but it strongly depends on channel asymmetry.

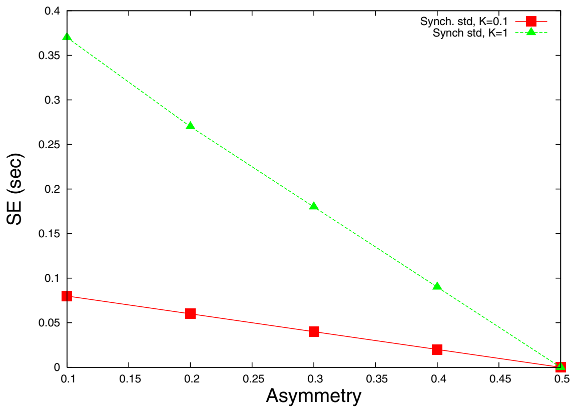
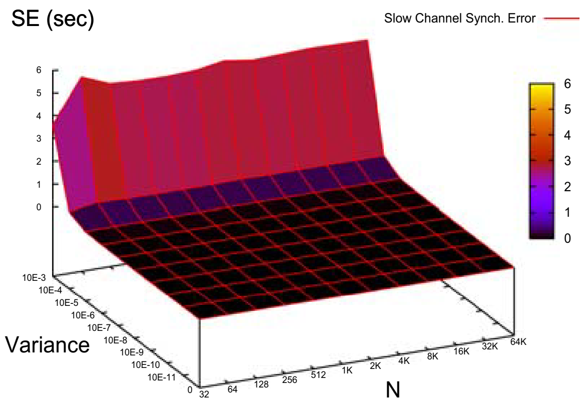
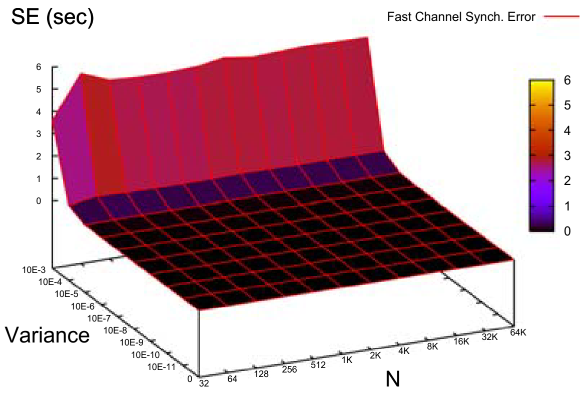


Fig. 3. Synchronization error varying channel asymmetry



(a) Slow Channels



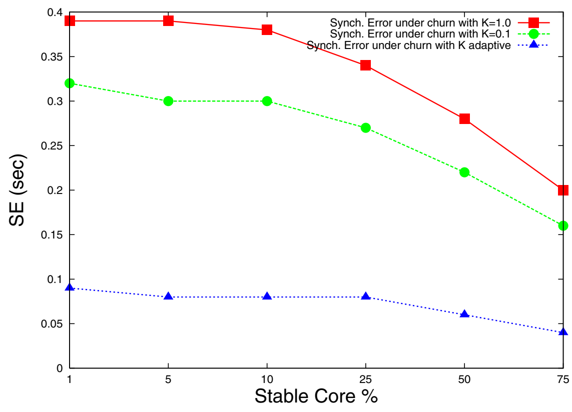
(b) Fast Channels

Fig. 4. Synchronization error for slow and fast Channels with a Gaussian asymmetry distribution

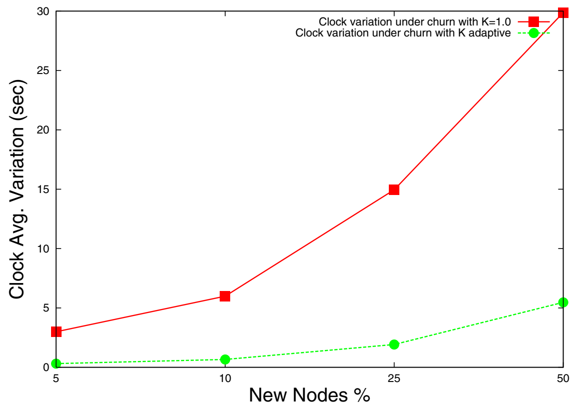
3.4 Dynamic Scenario with Symmetric Channels Results

Assuming as a deployment scenario a dynamic network with symmetric communication channels, we investigate how the dynamicity impacts on the synchronization error within which clocks synchronize, as well as on the stability of the clock value.

Synchronization Error. First we evaluated the resilience of our solution with respect to a continuous addition/removal of nodes. In this test we built a system with $64K$ nodes and considered a fixed *core* made up of nodes that remain in the system for the whole simulation. Dynamics is modeled substituting 1% of the system at each time unit. Then we evaluated how standard deviation of clocks residing in these nodes varies when the remaining part of the system keeps changing. The evaluation was done for two extreme values of the coupling factor K (*i.e.* $K = 0.1$ and $K = 1$), and also using an adaptive K value. Curves reported in Figure 5(a) show that the core size has a relevant impact on synchronization error as long as we consider a fixed K value. Intuitively, the larger is the core, the less nodes pertaining to it are prone to change their clock due to reads



(a) Synchronization error dependency on core network size and K value.



(b) Stability versus the number of new injected nodes and K value.

Fig. 5. System behavior under dynamics

done on newly joined nodes. In this case, by adopting a small value for the coupling factor, nodes belonging to the fixed core are more resilient to node dynamics. More interesting is the behavior of the system when we adopt the adaptive K strategy. In this case, new nodes enter the system using a large K value and therefore rapidly absorb timing information from nodes in the core, while these are slightly perturbed.

Stability. Figure 5(b) shows the stability of the system to a perturbation caused by the injection of a huge number of new nodes. In order to better show this behavior we introduced during the simulation in a network made up of $64K$ nodes, all with clocks synchronized on a specific value, a set of new nodes (expressed in the graph as a percentage of the original network size). Newly injected nodes start with a clock value that is distant 60 seconds from the synchronization value of nodes in the original system. The plot shows how the synchronization value varies from the original one (the distance between the synchronization values is reported in seconds on the y axis) when the new network converges again. If we assume $K = 1$ the system is prone to huge synchronization value variation, that, intuitively, is larger if a larger number of nodes is injected. However, the introduction of K adaptive mechanism drastically reduces this undesired behavior, limiting the variation of synchronization value, even when the amount of nodes injected is close to 50% of the original system.

These results justify the introduction of an age-based adaptive mechanism for the coupling factor value tuning as an effective solution to improve the stability of systems in face of dynamism.

4 Related Work

We can divide clock synchronization algorithms in two main classes: deterministic and probabilistic. Deterministic clock synchronization algorithms [9,12,13,14,15,18,19,20] guarantee strict properties on the accuracy of the synchronization but assumes that a known bound on message transfer delays exists. Lamport in [12] defines a distributed algorithm for synchronizing a system of logical clocks which can be used to totally order events, specializes this algorithm to synchronize physical clocks, and derives a bound on how far out of synchrony the clocks can become. Following works of Lamport and Melliar-Smith analyze the problem of clock synchronization in presence of faults, defining Byzantine clock synchronization [13,14]. Some deterministic solutions, such as those proposed in [7,8,14,17], prove that, when up to F reference time servers can suffer arbitrary failures, at least 2F+1 reference time servers are necessary for achieving clock synchronization. In this case, these solutions can be fault-tolerant also for Byzantine faults. Currently, we do not analyze byzantine-tolerant behavior of our solution. The deterministic approach, normally tuned to cope with the worst case scenario, assures a bounded accuracy in LAN environments but loses its significance in WAN environments where messages can suffer high and unpredictable variations in transmission delays. Several works of Dolev et al. [7,8,9,10] propose and analyze several decentralized synchronization protocols applicable for WAN but that require a clique-based interconnecting topology, which is hardly scalable with a large number of nodes.

Clock synchronization algorithms based on a probabilistic approach were proposed in [11,22]. The basic idea is to follow a master-slave pattern and synchronize clocks in the presence of unbounded communication delays by using a probabilistic remote clock reading procedure. Each node makes several attempts to read a remote clock and, after each attempt, calculates the maximum error. By retrying often enough, a node can read the other clock to any required precision with a probability as close to 1 as desired. This implies that the overhead imposed by the synchronization algorithm and the probability of loss of synchronization increases when the synchronization error is reduced. The master-slave approach and the execution of several attempts are basic building blocks of the most popular clock synchronization protocol for WAN settings: NTP [23,24]. NTP works in a static and manually-configured hierarchical topology. In this hierarchy, the primary time servers are directly connected to an external reference time source (GPS, atomic clock, etc.) and are the roots of the spanning tree used to diffuse clock values. Secondary time servers are represented by inner nodes in the hierarchy, and they synchronize their clock communicating with one or more primary time servers. Clients of the system resides in the hierarchy leaves. NTP requires complex analysis on samples, support from kernel and a controlled network between primary and secondary servers. A work proposing solutions close to NTP is CesiumSpray [25] that is based on a hierarchy composed by a WAN of LANs where in each LAN at least a node has a GPS receiver. This node acts as leader and "sprays" inside the local network its clock. These solutions require static configuration and the presence of some nodes directly connected with a external time reference in order to obtain external time synchronization.

A probabilistic solution based on a gossip-based protocol to achieve external clock synchronization is proposed in [16]. The presence of a source node perfectly synchronized with real-time clock is assumed. Each node uses a peer sampling service to select another node in the network and to exchange timing information with. If the time read from the contacted node is of higher quality than its own time (e.g. the contacted node is the source), then the reading node will adopt the clock setting of the other one. The quality of timing information is evaluated using a dispersion metric like the one provided by NTP.

5 Concluding Remarks

Clock synchronization for distributed systems is a fundamental problem that has been widely treated in the literature. Today's large scale distributed applications, deployed on WANs like Internet, pose new issues that are hardly addressed by existing solutions. These systems thus require the development of new approaches able to reach satisfying level of synchronization while providing the desired level of scalability.

In this paper we proposed a novel algorithm for clock synchronization in large scale dynamic systems in absence of external clock sources. Our algorithm stems from the work on coupled oscillators developed by Kuramoto [3], adequately adapted to our purposes. Through theoretical analysis backed up by an experimental study based on simulations we showed that our solution is able to converge and synchronize clocks in systems ranging from very small to very large sizes, achieving small synchronization errors that strictly depend on the quality of links used for communication (with respect to delay and symmetry). Our solution, thanks to the employment of an adaptable coupling factor, is also shown to be resilient to network dynamics, *i.e.* to the continuous arrival and departure of nodes in the system.

The approach to clock synchronization presented in this paper showed interesting properties, but is nevertheless susceptible to further improvements. There are mainly four points we plan to address in the next future. The K adaptive strategy employed in the current proposal (based on an exponentially decaying value) is only one of various possible solutions; we therefore plan to further investigate this aspect to, possibly, identify the best strategy to adapt K at run-time. Moreover, the adaptive mechanism could also be applied to ΔT: the basic idea is that stable nodes, whose clocks have possibly converged to a stable value, should "delay" further synchronization adopting larger values for ΔT. System adaptiveness can be also pushed one more step ahead, trying to rearrange node links in the interconnecting application-level network to favor neighbors connected to links that are more symmetric and that will therefore induce lower errors in clock readings. Finally, we plan to quickly implement a prototypical version of our algorithm to move our tests to the realistic testbed offered by PlanetLab. To this end, initial empirical evaluations [28] performed in our labs on a mid sized LAN confirm what predicted by the simulation presented in this paper.

References

1. Object Management Group. Data distribution service for real-time systems specication v1.2, ptc/2006-04-09
2. Winfree, A.T.: J. Theoret. Biol. 16, 15 (1967)
3. Kuramoto, Y.: Chemical oscillations, waves and turbulence, ch. 5. Springer, Berlin (1984)
4. Strogatz, S.H., Mirollo, R.E.: Phase-locking and critical phenomena in lattices of coupled nonlinear oscillators with random intrinsic frequencies. Physica D 31, 143–168 (1988)
5. Satoh, K.: Computer Experiment on the Cooperative Behavior of a Network of Interacting Nonlinear Oscillators. J. Phys. Soc. Jpn. 58, 2010 (1989)
6. Matthews, P.C., Mirollo, R.E., Strogatz, S.H.: Dynamics of a large system of coupled nonlinear oscillators. Physica D 52, 293 (1991)
7. Daliot, A., Dolev, D., Parnas, H.: Linear Time Byzantine Self-Stabilizing Clock Synchronization, Technical Report TR2003-89, Schools of Engineering and Computer Science, The Hebrew University of Jerusalem (December 2003)
8. Daliot, A., Dolev, D., Parnas, H.: Self-Stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks. In: Proc. Of the Sixth Symposium on Self-Stabilizing Systems, pp. 32–48 (2003)
9. Dolev, S.: Possible and Impossible Self-Stabilizing Digital Clock Synchronization in General Graph. Journal of Real-Time Systems 12(1), 95–107 (1997)
10. Herman, T., Ghosh, S.: Stabilizing Phase-Clock. Information Processing Letters 5(6), 585–598 (1994)
11. Cristian, F.: A probabilistic approach to distributed clock synchronization. Distributed Computing 3, 146–158 (1989)
12. Lamport, L.: Time, clocks and ordering of events in a distributed system. Commun ACM 21(7), 558–565 (1978)
13. Lamport, L., Melliar-Smith, P.M.: Byzantine clock synchronization. In: Proc. 3rd Ann. ACM Symp. Principles of Distributed Computing, pp. 68–74. ACM Press, New York (1984)
14. Lamport, L., Melliar-Smith, P.M.: Synchronizing clocks in the presence of faults. Journal of the ACM 32(1), 525278 (1985)
15. Lundelius-Welch, J., Lynch, N.: A new fault-tolerant algorithm for clock synchronization. In: Proc. 3rd Ann. ACM Symp. Principles of Distrib. Computing, pp. 75–88 (1984)
16. Iwanicki, K., van Steen, M., Voulgaris, S.: Gossip-based Synchronization for Large Scale Decentralized Systems (2006)
17. Cristian, F., Fetzer, C.: Integrating Internal and External Clock Synchronization. Journal of Real Time Systems 12(2) (March 1997)
18. Cristian, F., Fetzer, C.: Lower bounds for convergence function based clock synchronization. In: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing, August 20-23, pp. 137–143 (1995)
19. Cristian, F., Aghili, H., Strong, R.: Clock synchronization in the presence of omission and performance faults, and processor joins. In: Proc. Int. Conf. Fault-Tolerant Computing, pp. 218–223 (1986)
20. Halpern, J., Simons, B., Strong, R.: Fault-tolerant clock synchronization. In: Proc. 3rd Ann. ACM Symp. Principles of Distrib. Computing, pp. 89–102 (1984)

21. Kopetz, H., Ochsenreiter, W.: Clock synchronization in distributed real-time systems. IEE Trans. Comput. 36(8), 933–940 (1987)
22. Arvind, K.: Probabilistic Clock Synchronization in Distributed Systems. IEEE Trans. on Parallel and Distrib. Systems 5(5) (May 1994)
23. Mills, D.L.: Network Time Protocol (Version 1) specification and implementation. Network Working Group Report RFC-1059. University of Delaware (July 1988)
24. Mills, D.L.: Network Time Protocol Version 4 Reference and Implementation Guide. Electrical and Computer Engineering Technical Report 06-06-1, University of Delaware, p.83 (June 2006)
25. Verissimo, P., Rodrigues, L., Casimiro, A.: CesiumSpray: a Precise and Accurate Global Time Service for Large-scale Systems. Journal of Real-Time Systems 12(3), 243–294 (1997)
26. Jelasity, M., Guerraoui, R., Kermarrec, A.-M., van Steen, M.: The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware (2004)
27. Baldoni, R., Marchetti, C., Virgillito, A.: Impact of WAN Channel Behavior on End-to-end Latency of Replication Protocols. In: Proceedings of European Dependable Computing Conference (2006)
28. Baldoni, R., Corsaro, A., Querzoni, L., Scipioni, S., Tucci-Piergiovanni, S.: An Adaptive Coupling-Based Algorithm for Internal Clock Synchronization of Large Scale Dynamic Systems, MidLab Technical Report (February 2007), http://www.dis.uniroma1.it/~midlab

Reviewing Amnesia Support in Database Recovery Protocols*

Rubén de Juan-Marín, Luis H. García-Muñoz,
J. Enrique Armendáriz-Íñigo, and Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{rjuan, lgarcia, armendariz, fmunyoz}@iti.upv.es

Abstract. Replication is used for providing highly available and fault-tolerant information systems, which are constructed on top of replication and recovery protocols. An important aspect when designing these systems is the failure model assumed. Replicated databases literature last trends consist in adopting the crash-recovery with partial amnesia failure model because in most cases it shortens the recovery times. But, despite the large use of such failure model we consider that most of these works do not handle accurately the amnesia phenomenon. Therefore, in this paper we survey some works, analyzing their amnesia support.

1 Introduction

Database replication has become a key factor in providing fault tolerance, high availability and for increasing the performance of information systems. On one hand, performance can be improved when clients access the closest replica to them [1,2,3], or by using load-balancing algorithms [4,5,6]. On the other hand, replicas may fail or may disconnect; therefore, fault tolerance and high availability are reached forwarding client requests to non-failed nodes in a transparent way.

Latest trends in full database replication techniques –managed by replication protocols [1,2,3,4,5,6]– make use of a Group Communication System (GCS for short) [7] as it is detailed in [8]. These GCSs offer different services to the systems built atop of them. They provide several communication primitives, such as the atomic broadcast [9] allowing a more efficient implementation of replication protocols. Moreover, GCSs make use and provide membership mechanisms. The membership service keeps track of the active and connected nodes. Hence, reporting changes on the system configuration (i.e. failure or join of a replica). Being useful for determining if the replicated database progress condition is fulfilled: a majority of alive replicas –primary partition [7]–.

An important aspect in replicated database systems is how they manage crash node occurrences –which degrade their performance, fault tolerance and high availability support– and node connections or reconnections in order to maintain their original support. These systems have a special component named *recovery component* which deals with these situations –in a coordinated way with the consistency management performed by the replication protocol–, and the way it handles them depends on the

* Work supported by FEDER and the Spanish MEC grant TIN2006-14738-C02.

adopted failure model. The most commonly used failure models in replicated databases are *fail-stop* and *crash-recovery with partial amnesia*, as defined in [10].

The first one makes replicated systems discard crashed replicas, substituting them with new ones. Thus, when connecting a new or recovering node to the replicated system, the recovery protocol must first transfer the whole database to the recovering node before becoming fully operational, but this is impractical for large databases.

In order to avoid such drawback, last replicated database proposals [11,12] have adopted the *crash-recovery with partial amnesia* failure model. In this case crashed nodes are not discarded. The replicated system waits for their reconnection in order to start the recovery process. In this case the recovery protocol transfers to each recovering node only the database information lost during its disconnection. Once the recovering node has updated all its lost information, it becomes a fully operational node. So, in this approach the recovery process does not need to transfer the whole database but only the subset lost by the recovering node, shortening the recovery process and diminishing its associated problems.

But when assuming this second failure model, another problem appears: the amnesia phenomenon [13]. In this case, the problem relies on the difficulty that some protocols have to establish the correct subset of information to transfer when recovering. It appears because in some protocols the last assumed state in the recovering node is not really the last one, since such recovering replica may have lost some information that other replicas propagated to it before crashing.

In this paper, we briefly describe some database recovery protocols proposed in the literature [11,12,14,15,16,17,18] and surveyed in [19] emphasizing how they face the amnesia phenomenon. This study also gives special importance to the database replication protocols they are designed for, because as it was depicted in [19], the recovery protocols are very dependant from the characteristics of the replication protocols used, and the information that these replication protocols store.

The rest of the paper is structured as follows. In Section 2 we detail some important GCS aspects from the recovery point of view that will be used later. Section 3 presents the amnesia phenomenon, how it is manifested and how it can become a problem. Section 4 outlines the recovery protocols, and how they support the amnesia phenomenon, while in Section 5 we categorize the analyzed recovery techniques commenting if they provide basic amnesia support, or how they can be improved to support it when they do not. Finally, section 6 concludes the paper.

2 Group Communication System Issues

Before surveying the recovery protocols and the replication protocols they are designed for, we will present some issues dealing with replication aspects that would be used later in the study.

First of all, it must be introduced the *group communication system*. GCS provides some communication primitives that are used by replication protocols to perform their work. Primitives can vary from a point-to-point communication to total order broadcast.

This GCS also provides a *membership monitor* which informs group members about membership events –node connections, disconnections, network partitions, etc.–

Additionally, some GCSs provide *virtual synchrony* [7,20] (or *view-synchronous multicast* according to [21]) since it ensures that all replicas have delivered the same sequence of messages before any replica fails or any replica is added. According to [7], the most relaxed property related to multicast delivery that provides virtual synchrony is *same view delivery*; i.e., that all destinations of each multicast deliver each message when they belong to the same *view* (a view change arises when one process fails or rejoins the group). Virtual synchrony provides a replicated work way that facilitates the implementation of recovery protocols.

3 The Amnesia Phenomenon

At the beginning, database replication systems adopted the fail-stop failure model [10]. The reasons for adopting this failure model were: (a), it was the failure model mainly used in distributed systems; (b), its simplicity. In fact, when a replica crashes it is not recovered but substituted by a new one –transferring to it the whole state–. Therefore, the system must not generate and maintain special information for recovery purposes.

However, assuming this failure model implies some drawbacks when the replicated state information to be transferred is large –a common situation in replicated databases. Hence, the larger the information to be transferred the longer it takes to make a replica become active. This will imply, in the replicated system, the following consequences (already commented in [13]):

– Longer periods with decreased fault tolerance support. Only fully updated replicas can be used to guarantee the correct and consistent state evolution in the replicated system.
– Higher times of unavailability if the replicated system does not fulfill the progress condition (i.e. systems based on primary partitions).

In order to avoid all the above presented issues in replicated databases, researchers have opted for assuming the crash-recovery with partial amnesia failure model [10]. In this case, when a crashed replica reconnects the system recovers it transferring only the state it has missed; thus, transferring less information and minimizing the previous issues.

With the assumption of this failure model, the system is forced to determine correctly the subset of information that must be transferred to the recovering node. If it is not correctly determined, the state reached in the recovered node can diverge from the real consistent replicated state, leading to an undesired situation. In [13,22] we have already described this situation naming it as amnesia phenomenon, which manifests at two different levels:

– *Transport level*, at this level, it implies that the replica does not remember *which messages have been received*. Actually, the amnesia implies that received messages non-persistently stored are lost when the node crashes, generating a problem when they belong to transactions that the replicated system has committed but which have not been already committed in the crashed node, because message delivery does not really imply correctly processed as demonstrated in [23].

– *Replication level*, the amnesia is manifested here in the fact that the node “forgets” *which were the really committed transactions*. Usually, the internal log used by the underlying databases can be used for solving this.

3.1 A Generic Solution

We have also proposed a generic solution for overcoming this in [13,22]. The proposed solution consists in forcing each replica to enqueue persistently the broadcast messages as soon as they are delivered, removing them from this queue as soon as they are correctly processed. Then, when a crashed node reconnects, before asking about its missed changes to an updated replica it will check its queue of received and not applied messages (i.e. a log-based solution [18]). Obviously, this is not the unique solution that can be adopted for solving this problem, being possible also to apply version-based techniques [15].

3.2 Amnesia Formalization

Before surveying the considered works, we will first formalize the amnesia problem. To do so, we consider a replicated database system, $N = \{n_1, n_2, ..., n_n\}$, compound by n replicas, being $n > 2$ (primary partition assumption [7]). It uses an eager update everywhere protocol based on a GCS which provides an atomic broadcast primitive for spreading messages and virtual synchrony. It also uses constant interaction, broadcasting each transaction in a single message.

In this system, we identify each installed view –working view– as $\mathcal{V}_x$, being x the view identifier. $T_x = \{T_{x,1}, T_{x,2}, ..., T_{x,m}\}$ are the transactions delivered (and not aborted in this view –aborted transactions are not considered because they are not relevant for recovering purposes–). As the system uses the atomic broadcast primitive [9] for spreading transactions, all alive nodes deliver the broadcast transactions in the same order, using this order at execution time. This order is being reflected by the second subindex.

$\forall\, n_y \in \mathcal{V}_x$ we denote as T^D_{x,n_y} the transactions subset of T_x really delivered to n_y and, respectively, T^C_{x,n_y} the transactions subset of T_x really committed in n_y; fulfilling $T^C_{x,n_y} \subseteq T^D_{x,n_y}$. Virtual synchrony [7] ensures that $T^D_{x,n_y} = T_x$. Transition views are represented as $\mathcal{V}_x \rightarrow \mathcal{V}_{x+1}$.

Then $\forall\, \mathcal{V}_i \rightarrow \mathcal{V}_{i+1}$ triggered by a node crash, it will be at least one node $n_l : n_l \in \mathcal{V}_i \setminus \mathcal{V}_{i+1}$.

Considering that $T_i = \{T_{i,1}, T_{i,2}, ..., T_{i,m}\}$ is the transactions set delivered and committed in the replicated system during $\mathcal{V}_i$, it can be assumed that $\forall\, n_k \in \mathcal{V}_i \cap \mathcal{V}_{i+1}$:

$$T_i = T^D_{i,n_k} = T^C_{i,n_k} = \{T_{i,1}, T_{i,2}, ..., T_{i,m}\}$$

While $\forall\, n_l \in \mathcal{V}_i \setminus \mathcal{V}_{i+1}$, due to [23] it might happen the following:

$T_i = T^D_{i,n_l} \neq T^C_{i,n_l}$, where:

$T^C_{i,n_l} = \{T_{i,1}, T_{i,2}, ..., T_{i,m-s}\}$, being $s \in \{0, .., m\}$

In spite of assuming that $s \in \{0, .., m\}$ for simplicity reasons in this paper, it is also possible sometimes that $s > m$ due to workload reasons.

When n_l reconnects to the system, it triggers a new view $\mathcal{V}_{i+x}$, being $x > 1$. Later, the system must update it through the recovery process, transferring to it its lost transactions, which are:

– Transactions *forgotten* from its last seen view, $\mathcal{V}_i$: $T^F_{i,n_l} = T_{i,m-s+1}, ..., T_{i,m}$
– Transactions *missed* during its disconnection: $T^M_{n_l} = T_{i+1} \cup ... \cup T_{i+x-1}$

Then, for solving the amnesia phenomenon –*forgotten state*– when recovering n_l the two following properties must be provided:

– *Prop. FS1*: n_l must remember its last committed transaction, $T_{i,m-s}$;
– *Prop. FS2*: the replicated system must maintain and provide a way for obtaining the transactions subset T^F_{i,n_l} or their associated updates.

Once this *forgotten state* has been updated in the recovering replica, the recovery protocol can start with the recovery process itself, transferring *missed* data: $T^M_{n_l}$.

Notice that our generic solution, outlined in Section 3.1 and presented in [13,22], fulfils both properties. This is due to the fact that the persisted queue contains the messages associated to T^F_{i,n_l}.

It exists another way for solving the amnesia phenomenon in n_l which consist in transferring to the recovering node the whole state. We denote it as property *Prop. CS1* being an alternative to the previous ones. Obviously, this solution –the one used when adopting the fail-stop failure model– is not interesting when talking about systems managing large states.

4 Considered Recovery Protocols

In this section we briefly describe the recovery protocols considered in this study, highlighting only the details that are important from the amnesia support point of view. When detailed, we also include for each recovery protocol some remarks for our study about the replication protocols to which they are associated. For other details we encourage readers to look at the original papers. Note that in most cases the replication and recovery protocols were originally described in different papers (or even no replication protocol was described). As a result, a solution for the amnesia problems was not the target of such papers.

For determining if these recovery protocols provide accurate amnesia support we will study if they fulfill either the two properties *FS* or the property *CS* presented in Section 3.2.

4.1 Protocols by Kemme, Bartoli and Babaoğlu

Five different recovery protocols for replicated databases are presented in [14]. All of them are proposed for database replication protocols sharing the following characteristics: update everywhere protocol –ROWAA approach [24]–, based on total order broadcast –without a terminating phase– propagating a message per transaction –constant interaction [8]– and virtual synchrony. Moreover, replicated data objects are tagged with version numbers. The provided correctness criterion is *one-copy-serializability*.

These recovery protocols fulfil the following issues:

1. *Single Site Recovery*. The recovering node first brings its own database into a consistent state. To do so the underlying database maintains a log of performed writes during the normal processing, storing the initial and resulting values for each changed data object. Then, once it reconnects it checks this log in order to store in the database the changes of committed transactions that were not already applied in the database.
2. *Data Transfer*. An operating site –recoverer node– must provide the current database state to recovering nodes. Different techniques can be used, from transferring the full database to transferring only the set of updated objects.
3. *Determination of a Synchronization Point*. If transaction processing is allowed during the recovery process it must be ensured that the recovering node will reflect the updates performed by these transactions. This synchronization process can be done in different ways but it depends strongly on the data transfer technique.

On the sequel we describe the recovery protocols proposed in [14], focusing on the data transfer and synchronization points.

Database State Transfer Checking Version Numbers. In this protocol global transaction identifiers are used, marking each data object with the identifier of the last transaction that updated it –allowing later the system to determine the information set to transfer in recovery processes–. The recovering node informs to the recoverer node about its *cover* transaction, (i.e. the transaction with the highest global identifier that successfully committed, $T_{i,m-s}$). Thus, the recoverer can determine the updates lost by the recovering node –information that must be transferred–, including the updates associated to $T^F_{n_l}$. The recovering node can easily determine its *cover* transaction by reviewing its single site recovery log.

The amnesia phenomenon is avoided with this replication protocol because the recovering node tells to the recoverer node which is its last real committed transaction, $T_{i,m-s}$. Thus, the recovery process transfers all data objects that were modified by transactions delivered between the last real committed transaction in the recovering node and the first transaction propagated after it become alive. These lost updates are transferred using a DT. In properties terms:

– *Prop. FS1*: It is fulfilled because the recovering node remembers its last really committed transaction.
– *Prop. FS2*: Data modified by T^F_{i,n_l} are marked with the associated transaction identifier, so they will be later transferred in the recovery process. Notice that it is possible that these data are included when recovering $T^M_{n_l}$ and not by T^F_{i,n_l}, because these have been modified also during the node disconnection.

This recovery protocol presents the drawback of scanning the entire database when checking for the database subset to transfer, then the following proposal was designed to overcome it.

Restricting the Set of Objects to Check. In order to avoid the full scan on the entire database, and with this the overhead and long locking time that it may cause, the use of a so-called "reconstruction table" is proposed. A record in this table consists of an object identifier and a global identifier informing about the last transaction that updated the object. Each update is recorded in the reconstruction table, unless all sites have successfully performed the update.

In contrast to the previously discussed protocol options, this one only needs to set a single lock on the entire database. Once the incremental data set to be transferred is determined, that lock is replaced by fine-grained object level locks on the respective data items.

This proposed optimization does not handle correctly the amnesia phenomenon. This is because the reconstruction table only is generated when there are failed nodes. Then, only those objects modified in a view with failed nodes are included in this table. It implies that if a failed node has not been able to process correctly a message delivered before crashing –in a view with no failed nodes–, when it reconnects it will have not applied the associated updates. And the reconstruction table will have not stored these updates because they have been performed in a view without failed replicas, being unable the recovery system to transfer the correct information set. Expressing it in properties terms:

– *Prop. FS1*: It is fulfilled because the recovering node remembers its last really committed transaction.
– *Prop. FS2*: It is not always fulfiled because data modified by T^F_{i,n_l} is only stored in the reconstruction table and marked with the associated transaction identifier if there are failed nodes. Therefore, the system will not have the T^F_{i,n_l} changes in the reconstruction table when a replica crashes in a replicated system where there were not any crashed node.

Therefore, this recovery protocol must be modified in order to support the amnesia phenomenon. One possibility consists in generating the recovery information in the reconstruction table either with or without the presence of failed nodes, ensuring always the *Prop. FS2*. Other possibility consists in using our proposed solution in [13] ensuring then always *Prop. FS2*.

Filtering the Log. In the previously discussed optimization, locking of non-relevant data is reduced, but locks on relevant data may still last long. To avoid locks, multiple versions of data can be used, e.g., the use of multi-version concurrency control, as in `PostgreSQL`, or `Oracle`. In that case, transactions can continue to update the database while earlier versions that have been missed by the recovering site are transferred to it.

This recovery technique must be combined with one of the previous ones, that will be used for generating the recovery information and determining the subset to transfer, for working. Therefore, its amnesia support depends on the support provided by the combined technique. Hence, if the last one –*Restricting the Set of Objects to Check*– is selected in its original description the amnesia phenomenon will not be managed accurately.

Lazy Data Transfer. Up to this point, all mentioned solutions use view changes as synchronization points. Then any recovering node enqueues all new broadcast transactions for applying them once it has recovered its lost views. This approach, despite being simple, has several drawbacks (detailed in [14]) leading the authors to decouple the synchronization point from the view change.

In this new approach any recovering site discards the messages delivered –instead of enqueuing them–. After the view change –triggered by its reconnection–, the recoverer site starts the transfer. When the transfer is about to complete, the recoverer and the recovering sites agree a delimiter transaction –one of the transactions broadcast in the new view– as synchronization point. Then, the recoverer site transfers all changes performed by transactions with lower identifier than the delimiter transaction one. Concurrently, the recovering site starts enqueuing transaction messages with greater identifier than the delimiter transaction one, for applying them once the data transfer is completed.

This recovery proposal differs from the others in the synchronization point, but depends on the previous ones for obtaining the recovery information. Then, it will support the amnesia phenomenon depending on the recovery information generation policy used.

4.2 Protocols by Holliday

The recovery protocols proposed by J. Holliday in [12], were designed for the replication protocols *Broadcast Writes*, *Delayed Broadcast* and *Single Broadcast* described in [25]. According to the classification in [26], these are eager update everywhere and non-voting protocols. Concurrency control is performed by the DBMS with Strict 2PL. These replication protocols make also use of a GCS which provides atomic broadcast, virtual synchrony and a membership monitor. They provide the *one-copy serializability* correctness criterion.

These replication protocols differ in the number of messages used for propagating transactions. *Single Broadcast* only spreads a message per transaction, *Delayed Broadcast* propagates two messages per transaction –writeset and commit messages–, while *Broadcast Writes* sends a message per write transaction operation –linear interaction–.

On the sequel we will summarize the recovery approaches presented in [12].

Single Broadcast Recovery. This recovery approach is designed for replication protocols which broadcast a single message per transaction as [3]. Another author criterion design is to avoid to transfer the whole database in the recovery process if possible, and the selected mechanism for doing so consists in reapplying in outdated nodes the messages that they have lost.

Therefore, this recovery protocol relies on a GCS which provides a log of delivered messages. If the GCS does not provide this log, the recovery protocol must designate some replicas as *loggers*. These *loggers* will have a log where they will store persistently the delivered messages, i.e., those notifying view changes and those broadcasting update transactions –keeping only those associated to committed transactions, deleting aborted ones–.

Then, when a node reconnects –a view change is triggered– it requests a *logger* to be brought up-to-date, informing about its last view –last view in which the recovering node was alive. Thus, the *logger* transfers to the outdated node the messages broadcast during the views it was crashed –maintaining their original order. Sometimes the *logger* will not have, due to log storing policies, all the messages necessary in the recovery, thus it will transfer the whole database. It must be remarked that as long as a node is being recovered the replicated system can not work, starting only when the recovery process has been completed.

This protocol does not support the amnesia phenomenon accurately, because when a crashed node reconnects, the system starts to transfer the messages broadcast in the views it was failed. Then, this information does not include messages delivered in the crashed node before crashing but not processed correctly. In properties terms:

- *Prop. FS1*: It is not fulfilled because the recovering node only remembers its last seen view, i.e., it does not maintain nor propagate the $T_{i,m-s}$ identifier.
- *Prop. FS2*: This property is not fulfilled because messages –recovery information– are stored by view. But it can be easily overcome using messages as basic recovery information unit.

One possibility for handling accurately the amnesia phenomenon in this recovery protocol would be to use message or transaction identifiers –which are equivalent in this replication protocol. Then the recovery protocol can be modified forcing each replica to mark which is its last really committed transaction. Therefore, when a replica reconnects after a crash, it can inform about its last committed transaction to the *Logger* –instead of using the identifier of its last seen view– for the recovery. Then both properties are ensured.

Another possibility would consist in giving the *Loggers* role to all the replicated system members. Therefore when a node reconnects, it will perform a local recovery step consisting in checking if some of the persisted messages in its local log have not been correctly processed, applying them in this step. In this case both properties are also ensured.

In both cases, notice that it is necessary that *Loggers* store persistently the broadcast messages even when there are no failed nodes. In the second approach, the messages that have been seen by all nodes –because there are not any failed nodes– can be removed from the log –of a replica– as soon as they are correctly processed in this replica. While, in the first approach, these messages can only be removed view per view. And the messages broadcast in a view where there were not failed nodes, only can be removed if in the subsequent view there are not any failed nodes –which is a non sense– or when the nodes whose crash triggered this view change are recovered and the system ensures that they have correctly processed all the messages broadcast when there were not failed nodes.

The second solution provides better recovery support as all replicated members are *Loggers*, and provides a more simple way for managing messages that have been seen by all nodes. Therefore, we encourage its use.

Delayed Broadcast Recovery. The *Delayed Broadcast* replication protocol decouples the writeset broadcast from the commit broadcast for any transaction –weak voting technique [27]–. This behavior raises some problems when recovery is being considered. It might happen that the recovering site was able to deliver the writeset for a particular transaction, but not its commit or rollback message. So, that writeset was lost when the site failed and should be retransmitted now by the recoverer site if its commit message was delivered whilst the recovering site was crashed. Two possible solutions for the problems caused by the writeset-commit decoupling are presented:

1. *Log Update Method*. In this approach, at each view change, *loggers* must examine their logs or the database state for determining if there exist on progress transactions in the nodes without failure. If there are, the *logger* must mark these transactions in order to copy their writeset message in the log associated to the view when their commit was broadcast. So, when a previously failed node rejoins to the group, the *logger* begins transferring writesets of in progress transactions when the node failed, following with messages of transactions originated and committed while the node was failed. The commit order is the same for all non-aborted transactions. The operations of the aborted transactions are not included in the log since their effects are undone in the nodes without failure.
2. *Augmented Broadcast Method*. This second method gives additional process for managing on-going transactions and requires a change in the lock policy for recovering nodes during the global recovery. The new replication protocol forces to include the writeset in the broadcast commit message for these transactions that have delivered the writeset in a previous view. The nodes that have already seen the first broadcast writeset message ignore the writes included in the commit message, and *loggers* store the augmented commit message. The existence of augmented messages obliges global recovery to change its lock policy as it is described in [12].

In this case, as the policy for determining the start point recovery is the same one as before –the identifier of the last view in which the crashed node was alive–, an accurate amnesia phenomenon support is not provided. Explained in properties terms:

– *Prop. FS1*: As before, it is not fulfilled because the recovering node only remembers its last seen view.
– *Prop. FS2*: This property is not fulfilled because messages –recovery information– are stored by view.

Therefore, the modifications proposed in the previous recovery protocol are also valid for this one. Anyway, it must be pointed out that in this case handling delivered messages correctly is more difficult because the system broadcasts two messages per transaction –*writeset* and *commit* or *rollback*– with its associated complexities.

If the second solution –all nodes are *Loggers*– is selected, when the recovering node performs the additional local recovery step –checking if some of the persisted messages in its local log have not been correctly processed for applying them– it must discard the messages belonging to transactions whose commit message is not also stored in the log. This is because these messages would be later applied in the *Global Recovery* process.

Broadcast Writes Recovery. The *Augmented Broadcast* global recovery method presented for the *Delayed Broadcast* replication protocol could be used also for the *Broadcast Writes* one –which broadcasts a message for each write operation, in other words linear interaction–. Then all writes must be attached to the commit message to be broadcast for on-going transactions, as it does the *Augmented Broadcast*. But in this recovery protocol *loggers* must take special care for removing the logged messages of aborted transactions due to deadlocks, in order to not reapply them in recovering nodes.

As the two other recovery protocols proposed by Hollyday it does not handle correctly the amnesia phenomenon problem, because the underlying mechanism for determining the recovery information set to transfer is the same one –to send the identifier of the last view seen by the crashed node–. In properties terms:

- *Prop. FS1*: As in two previous ones, it is not fulfilled because the recovering node only remembers its last seen view.
- *Prop. FS2*: This property is not fulfilled because messages –recovery information– are stored by view.

Anyway, the proposed solutions for the previous recovery protocol will also work for this one.

4.3 Parallel Recovery by Jiménez, Patiño and Alonso

In [11] the authors presented a recovery protocol whose main goal was to avoid stopping the replicated system work when performing recovery processes.

The replication protocol for which it was designed used a GCS that provided strong virtual synchrony, reliable multicast and a membership monitor. The replicated database was divided into disjoint partitions,and the system forced transactions to access only single partitions. Each partition had a master site –which processed the transactions accessing this partition– and the rest of replicas worked as backups –which only applied updates–, therefore it is a passive replication protocol per partition. And transactions are broadcast using only one message –constant interaction–. The transactional system supports Strict 2PL, providing *one-copy serializability*.

Each node has a log –one per partition– which contains the committed updates in the same order they were applied. Updates are only logged once their commit is confirmed. When a crashed node reconnects to the system it informs about the LSN –*log sequence number*, a global number– of its last committed transaction on each partition. Then the selected recoverer for each partition will collect and transfer from its log the set of messages needed to recover this partition in the outdated node. In order to limit the recovery duration –interesting for long failure times– some form of checkpointing is assumed. Therefore, if it is necessary, the recovering site will first receive a recent checkpoint of the database and later can start applying messages from this checkpoint.

The combination of these two techniques, or the use of the *LSN* of the last committed transaction in the node being recovered allows the protocol to overcome the amnesia problem. The problem of this solution depends on the way in which the checkpoint process is performed, because if the whole data state is transferred the benefits of adopting the crash-recovery with partial amnesia failure model are lost. Expressed in properties terms:

- *Prop. FS1*: It is fulfilled with the use of *LSN*.
- *Prop. FS2*: This property is fulfilled because nodes store committed updates and combines this with a checkpointing technique when necessary.

4.4 The COLUP Recovery Protocol

A configurable eager/lazy replication protocol with a lazy recovery protocol is proposed in [17]. The replication protocol can be categorized as an update everywhere approach with voting technique, using constant interaction. This protocol defined and provided its own correctness guarantees: *transaction* and *checkout* consistency. These correctness guarantees are somewhat equivalent in some circumstances to *snapshot isolation* and *read committed* respectively.

In this replication protocol each data object is owned by the replica where it was created. For any object, a set of nodes will maintain synchronous copies, while other replicas constitute the set of asynchronous copies. In these last nodes object updates will be eventually received, once they have been committed in synchronous replicas. The owner is responsible of managing object accesses and coordinating the propagation of their last versions.

Conflict transactions are solved in the processing node in an optimistic way, using object versions. To do so, for each accessed object –for those the node does not have a synchronous copy– it calculates the probability of having an outdated version. If the obtained value is higher than an established threshold the node assumes that its object version is obsolete, obtaining from the owner node the last version. Later, in the commit phase it checks for possible conflicts. Aborting the transaction if it has read obsolete values that were updated by other concurrently committed transactions.

In a node crash the ownership of its objects is assumed by an alive and synchronized replica. Then, alive nodes inform the new owner about *previous grants* conceded to these objects by the previous owner. Thus, the new owner can process the requests as if it was the original owner node of the object.

When a node recovers from a failure, it sends a message to the node that managed its owned objects in order to synchronize the activity in both nodes. In this process, the recovering node updates in a version-based way the state either of its owned objects and the objects for which it is a synchronized replica. During this process, the recovering node may receive requests for objects that were updated during the failure interval. In order to handle this situation, the recovering node must consider each object of which it is owner like an asynchronous replica until it is updated by a synchronous replica.

This recovery protocol provides accurate amnesia support because as soon as a node reconnects it starts to obtain the last state of its owned and synchronized objects in a version-based way. The objects that are maintained in this replica asynchronously are updated using the basic mechanism provided by the replication protocol. This protocol fulfils the *Prop. CS1*, because all the state is transferred: synchronized objects are transferred inmediately in the recovering process, and non-synchronized ones are updated using the replication mechanism. Therefore, the other properties are not needed. It must be noticed that in spite of adopting the crash-recovery with partial amnesia failure model this recovery protocol transfers the whole state instead of sending only the missed information during the disconnection period.

4.5 CLOB: Short-Term Failure Recovery

CLOB (Configurable LOgging for Broadcast protocols) described in [18] is defined as a framework for reliable broadcast protocols that are used as a basis for database replication. Its aim is to log messages in the broadcast protocol core, providing with this automatic recovery for short-term failures, but discarding the log and using a version-based recovery protocol (e.g. [15]) for long-term outages.

In order to do so the recovery protocol has two logs: one for missed messages, another for received messages. In the first one, each node stores any message it delivers when there are failed nodes, maintaining them as long as there is any failed node that has not received them. In the second one, each node stores any received message, removing it as soon as it is correctly processed. So, when a crashed node reconnects –and the system uses the log recovery–, it first checks the log of received messages in order to process its last received messages that were not correctly processed before crashing. Later, it asks for its missed messages, and applies them.

Notice that if the outage period exceeds a given threshold, the reliable broadcast service will notify the replication protocol about that, and the logs will not be used.

The CLOB recovery protocol manages accurately the amnesia phenomenon because it considers a persistent log where each replica stores its delivered messages as soon as they are received. And these messages are only deleted once they are correctly processed. Then, when a crashed node reconnects, only needs to check this log and reapply the messages it contains. Talking about properties:

- *Prop. FS1*: It is fulfilled in an indirect way. All messages maintained in the queue represent delivered transactions non correctly processed, so instead of knowing its last really committed transaction it has the T^F_{i,n_l}.
- *Prop. FS2*: As it has been said above, each node stores persistently its own T^F_{i,n_l}.

4.6 Protocol by Armendáriz

In [16] three replication protocols are considered –*BRP*, *ERP* and *TORPE* –, and a recovery protocol that can be applied on *ERP* and *TORPE* is proposed. These two replication protocols are categorized for being eager update everywhere and sending a constant number of messages per transaction. They make use of a GCS which provides reliable broadcast, a membership monitor and virtual synchrony. The correctness guarantees provided by these protocols were *one-copy serializability*, provided thanks to the use of underlying DBMS which ensured serializability.

The main idea for the recovery protocol proposed in [16] is to store in a database table –in all alive replicas– the identifiers of objects modified when there are failed nodes, grouping them per views. Then, when a failed node reconnects, it informs about the last view in which it was alive. Later, a recoverer node transfers to the recovering node the identifiers of modified objects during its disconnection, and later transfers their values.

The recovery protocol proposed by Armendáriz for the replication protocols ERP and TORPE can not manage accurately the amnesia problem. In this case, the problem resides in the fact that this recovery protocol assumes that any delivered message is correctly processed, but this assumption, as demonstrated in [23], is not correct. So, all generated recovery information does not contain all the information that would be needed for supporting amnesia. Expressing all this in properties terms:

- *Prop. FS1*: It is not fulfilled because the recovering node only remembers its last seen view.
- *Prop. FS2*: This property is not fulfilled because the recovery information is grouped by view. And either it presents the problem of being generated only when there are failed nodes.

In [28] it is provided amnesia support to this recovery protocol. The adopted solution is the same one as we propose in Section 3.1, to log persistently the delivered messages.

5 Amnesia Support Recovery Observations

We have seen in the study how a correct amnesia support depends on the combination of an adequate recovery information generation policy and an accurate way for notifying the last really committed changes in the node that must be recovered.

On the sequel, we will present some observations obtained from the performed study. These observations are grouped first by the used technique –version-based or log-based–, and secondly by the granularity used for managing the recovery information.

We will not consider the recovery solution consisting in transferring the whole state, because the original goal of adopting the crash-recovery with partial amnesia failure model in replicated systems is to avoid its use.

5.1 Version-Based Techniques

Version-based recovery protocols can overcome this problem in different ways, depending on the basic way used for performing the recovery processes.

Transaction identifier. The first one will consist in storing for each object the identifier of the last transaction that modified it. But, this must be done even if there are not failed nodes as it does the *Database State Transfer Checking Version Numbers* presented in [14], because if it is not done the amnesia support is not provided as it happens with *Restricting the Set of Objects to Check* presented also in [14]. Thus, in this case the recovering node only has to inform the recoverer node about the identifier of its last committed transaction. Therefore, properties *Prop. FS1* and *Prop. FS2* are ensured.

An alternative for this strategy will be to combine it with our amnesia generic solution approach described in Section 3. In this case it would not be necessary to generate this information even when there are not failed nodes. And, then this approach does not need the transaction granularity being enough with the view identifier granularity. It is due to the fact that in this case each replica maitains its own T^F_{i,n_l}, being only necessary to inform the recoverer node about the last seen view in the recovering node.

View identifier. Another possibility is to store for each object the identifier of the last view in which it was modified. The problem of this solution is that the recovery protocols that follow this approach start the recovery process from the first view lost by the recovering node, being impossible then to solve the amnesia problem, associated to the *forgotten state* –T^F_{i,n_l}– because even if *Prop. FS1* is ensured, *Prop. FS2* is not ensured. It happens in *Protocol by Armendáriz* [16]. This can be solved as follows:

- One option for overcoming this would consist in including in the transfer recovery process the changes performed in the last view seen by the recovering node. So, this solution forces the system to generate recovery information even when there are not failed nodes. But, this approach presents some drawbacks. On one hand, it forces to transfer all the performed changes in a view –most of which will have been already seen by the recovering node– for solving the amnesia problem that will affect usually a very small subset of changes done in such view. On the other hand, it is possible that in very special cases transferring only the changes done in the last view seen by the recovering node is not enough for solving the amnesia problem (e.g. a sequence of very short views in time terms).
- Discarding the previous option, another strategy will consist in combining this strategy with our generic approach –using in each replica a persistent log of delivered messages– as it is done in [28], fulfilling then the properties *Prop. FS1* and *Prop. FS2*. In this case, it is not necessary for the version-based strategy to generate information when there are not failed nodes, because it is already maintained in the queue.

5.2 Log-Based Techniques

In these techniques, recovery protocols use as recovery information the broadcast messages during the replication work. Therefore, the only way for solving the amnesia problem is to maintain in the system the messages that can be affected by the amnesia problem.

Transaction identifier. In this technique, stored messages –all replicas store messages– are not grouped by views, then when a crashed node reconnects it informs about the message corresponding to its last committed transaction. Then, the recoverer node sends to the recovering node the set of messages it has not correctly processed and it has lost. Notice that this policy will overcome the amnesia phenomenon in all cases, only if logs store messages even when there are not failed nodes. If this behavior is not provided the *Prop. FS2* is not ensured when a replicated system transits from a view where all replicas were alive to another where there are failed nodes.

An important aspect of this technique is when messages or updates are stored in the log. If messages are persisted as soon as they are delivered, crashed nodes will have at recovering time the messages they have delivered but not processed correctly –those associated to T^F_{i,n_l}–. Then, they do not have to ask updated replicas for these messages, only for those they have not seen. On the contrary, if messages –or updates– only are logged when they are really committed, crashed nodes will not have the messages necessary for overcoming the amnesia problem at recovering time. So, in this case the information for solving the amnesia phenomenon must be looked for in the recoverer replica.

This is the case of the *Parallel Recovery by Jiménez, Patiño and Alonso* [11] protocol. This protocol also combines this technique with checkpointing for log shortening reasons. It must be noticed that this protocol stores updates once they are committed and not at delivery time, so crashed replicas must ask updated replicas for messages delivered but not correctly processed.

View identifier. In this strategy broadcast messages are stored when they are delivered –in the same order delivery– being grouped by views –when there are crashed nodes–. Then, when a crashed node reconnects it informs to the system about its last seen view. At this point, the system starts to send to the recovering node the messages broadcast during the view it was crashed. Therefore, the amnesia problem is not solved as it occurs in all recovery protocols proposed in [12], because it will not contain messages seen by the crashed node but non correctly applied, in other words the recovery process does not transfer the messages corresponding to the transactions set T^F_{i,n_l}. In fact, neither *Prop. FS1* nor *Prop. FS2* are ensured. For solving this problem, two different approaches can be adopted:

- A first proposal for avoiding the amnesia problem in this technique can consist in transferring in the recovery process the messages broadcast during the last view where the crashed node was alive. Then, this solution needs to store broadcast messages even if there are not failed nodes. But, it can be optimized if the recovering node informs about the identifier of the message associated to its last correctly processed transaction. Moreover, it must be noticed that if all nodes store broadcast messages the own crashed node will contain the messages it has received and not correctly applied, obtaining then the second approach.
- The second one consists in applying our proposed generic solution, that in fact is the solution already applied in [23,18]. In [23], authors proposed the "*successful delivery*" approach. A successfully delivered message implies that it has been correctly processed. Therefore, they proposed that the used GCS has to deliver the same message to a replica until it is successfully delivered in this replica. In [18], each node stores persistently all its delivered messages, being only removed when they are correctly processed. Obviously, if there are failed nodes, correctly processed messages are not removed but maintained in another log for recovering failed nodes during this view.

6 Conclusions

In this survey we have analyzed how some recovery solutions for replicated databases, which have adopted the crash-recovery with partial amnesia failure model –in order to avoid to transfer the whole database–, manage the introduced amnesia phenomenon problem.

This problem appears because some works assume that all delivered messages are correctly processed, fact that as it is demonstrated in [23] is not true. Then, in most cases their provided recovery solutions do not handle correctly this problem. Among the studied papers only the recovery protocols proposed in [11,17,18] and two of [14] manage accurately this problem.

Moreover, for those studied recovery protocols which do not provide accurate amnesia support we have proposed solutions for overcoming this situation.

Later, we have categorized the analyzed recovery techniques commenting if they provide accurate amnesia support, and how they can be improved to support when they do not in their original definition.

References

1. Muñoz-Escoí, F.D., Pla-Civera, J., Ruiz-Fuertes, M.I., Irún-Briz, L., Decker, H., Armendáriz-Iñigo, J.E., de Mendívil, J.R.G.: Managing transaction conflicts in middleware-based database replication architectures. In: SRDS, pp. 401–410. IEEE-CS, Los Alamitos (2006)
2. Patiño-Martínez, M., Jiménez-Peris, R., Kemme, B., Alonso, G.: Middle-r: Consistent database replication at the middleware level. ACM Trans. Comput. Syst. 23, 375–423 (2005)
3. Lin, Y., Kemme, B., Patiño-Martínez, M., Jiménez-Peris, R.: Middleware based data replication providing snapshot isolation. In: Ozcan, F. (ed.) SIGMOD Conf. pp. 419–430. ACM, New York (2005)
4. Plattner, C., Alonso, G.: Ganymed: Scalable replication for transactional web applications. In: Jacobsen, H.A. (ed.) Middleware 2004. LNCS, vol. 3231, pp. 155–174. Springer, Heidelberg (2004)
5. Elnikety, S., Dropsho, S., Zwaenepoel, W.: Tashkent+: Memory-aware load balancing and update filtering in replicated databases. In: Proc. EuroSys 2007, pp. 399–412 (2007)
6. Amza, C., Cox, A.L., Zwaenepoel, W.: A comparative evaluation of transparent scaling techniques for dynamic content servers. In: ICDE, pp. 230–241. IEEE Computer Society Press, Los Alamitos (2005)
7. Chockler, G.V., Keidar, I., Vitenberg, R.: Group communication specifications: A comprehensive study. ACM Computing Surveys 4, 1–43 (2001)
8. Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G.: Understanding replication in databases and distributed systems. In: ICDCS, pp. 464–474. IEEE Computer Society Press, Los Alamitos (2000)
9. Hadzilacos, V., Toueg, S.: Fault-tolerant broadcasts and related problems. In: Mullender, S. (ed.) Distributed Systems, pp. 97–145. ACM Press, New York (1993)
10. Cristian, F.: Understanding fault-tolerant distributed systems. Communications of the ACM 34, 56–78 (1991)
11. Jiménez, R., Patiño, M., Alonso, G.: An algorithm for non-intrusive, parallel recovery of replicated data and its correctness. In: SRDS, pp. 150–159 (2002)
12. Holliday, J.: Replicated database recovery using multicast communication. In: NCA, pp. 104–107. IEEE Computer Society Press, Los Alamitos (2001)
13. de Juan-Marín, R., Irún-Briz, L., Muñoz-Escoí, F.D.: Supporting amnesia in log-based recovery protocols. In: EATIS. Euro American Conference on Telematics and Information Systems, Faro, Portugal (2007)
14. Kemme, B., Bartoli, A., Babaoğlu, O.: Online reconfiguration in replicated databases based on group communication. In: Intl.Conf.on Dependable Systems and Networks, Washington, DC, USA, pp. 117–130 (2001)
15. Castro, F., Irún, L., García, F., Muñoz, F.: FOBr: A version-based recovery protocol for replicated databases. In: 13th Euromicro PDP, Lugano, Sw, pp. 306–313 (2005)
16. Armendáriz, J.E.: Design and Implementation of Database Replication Protocols in the MADIS Architecture. PhD thesis, Univ. Pública de Navarra, Pamplona, Spain (2006)
17. Irún, L., Castro, F., García, F., Calero, A., Muñoz, F.: Lazy recovery in a hybrid database replication protocol. In: XII Jornadas de Concurrencia y Sistemas Distribuidos, pp. 295–307 (2004)
18. Castro, F., Esparza, J., Ruiz, M., Irún, L., Decker, H., Muñoz, F.: CLOB: Communication support for efficient replicated database recovery. In: 13th Euromicro PDP, Lugano, Sw, pp. 314–321. IEEE Computer Society, Los Alamitos (2005)
19. García-Muñoz, L.H., Armendáriz-Íñigo, J.E., Decker, H., Muñoz-Escoí, F.D.: Recovery protocols for replicated databases - a survey. In: Workshop FINA-07, in the AINA-07 Conference, pp. 220–227. IEEE-CS Press, Los Alamitos (2007)

20. Birman, K., Joseph, T.: Exploiting virtual synchrony in distributed systems. In: 11th ACM Symposium on Operating Systems Principles, pp. 123–138. ACM Press, New York (1987)
21. Guerraoui, R., Schiper, A.: Software-based replication for fault tolerance. IEEE Computer 30, 68–74 (1997)
22. de Juan-Marín, R., Irún-Briz, L., Muñoz-Escoí, F.D.: Recovery strategies for linear replication. In: Guo, M., Yang, L.T., Di Martino, B., Zima, H.P., Dongarra, J., Tang, F. (eds.) ISPA 2006. LNCS, vol. 4330, pp. 710–723. Springer, Heidelberg (2006)
23. Wiesmann, M., Schiper, A.: Beyond 1-Safety and 2-Safety for replicated databases: Group-Safety. In: 9th International Conference on Extending Database Technology, pp. 165–182 (2004)
24. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: ACM SIGMOD International Conference on Management of Data, pp. 173–182. ACM Press, New York (1996)
25. Agrawal, D., Alonso, G., El Abbadi, A., Stanoi, I.: Exploiting atomic broadcast in replicated databases. In: Lengauer, C., Griebl, M., Gorlatch, S. (eds.) Euro-Par 1997. LNCS, vol. 1300, pp. 496–503. Springer, Heidelberg (1997)
26. Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G.: Database replication techniques: a three parameter classification. In: SRDS, pp. 206–215 (2000)
27. Wiesmann, M., Schiper, A.: Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng. 17, 551–566 (2005)
28. García-Muñoz, L.H., de Juan-Marín, R., Armendáriz, J.E., Muñoz-Escoí, F.D.: Improving Recovery in Weak-Voting Data Replication. In: 7th International Symposium on Advanced Parallel Processing Technologie, Guangzhou, China (2007)

The Conceptualization of a Configurable Multi-party Multi-message Request-Reply Conversation

Nataliya Mulyar[1], Lachlan Aldred[2], and Wil M.P. van der Aalst[1,2]

[1] Department of Technology Management, Eindhoven University of Technology
GPO Box 513, NL5600 MB Eindhoven, The Netherlands
{n.mulyar, w.m.p.v.d.aalst}@tue.nl

[2] Faculty of Information Technology, Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
{l.aldred}@qut.edu.au

Abstract. Organizations, to function effectively and expand their boundaries, require a deep insight into both process orchestration and choreography of cross-organization business processes. The set of requirements for service interactions is significant, and has not yet been sufficiently refined. Service Interaction Patterns studies by Barros et al. demonstrate this point. However, they overlook some important aspects of service interaction of bilateral and multilateral nature. Furthermore, the definition of these patterns are not precise due to the absence of a formal semantics. In this paper, we analyze and present a set of patterns formed around the subset of patterns documented by Barros et al. concerned with Request-Reply interactions, and extend these ideas to cover multiple parties and multiple messages. We concentrate on the interaction between multiple parties, and analyze issues of a non-guaranteed response and different aspects of message handling. We propose one configurable, formally defined, conceptual model to describe and analyze options and variants of request-reply patterns. Furthermore, we propose a graphical notation to depict every pattern variant, and formalize the semantics by means of Coloured Petri Nets. In addition, we apply this pattern family to evaluate WS-BPEL v2.0 and check how selected pattern variants can be operationalized in Oracle BPEL PM.

1 Introduction

It has been several years since Service-Oriented Architectures (SOAs) started gaining enormous popularity within organizations aiming to extend their boundaries by integrating software applications and external services into their business processes. To coordinate the interaction between service providers and consumers a set of standards and technologies were proposed which contributed in evolution of the Web-services paradigm. Standards like SOAP [1], WSDL [2], UDDI [3], etc. were proposed to interconnect independently developed web-services. A number of standardization proposals (XLang, BPML, and WSCI) [4,5,6,7] were discontinued, however they have served as a basis for an ongoing standardization initiative: the Business Process Execution Language for Web-Services (BPEL4WS, BPEL, WSBPEL) [8]. The developed technologies successfully handle simple interaction scenarios, however when it comes to interactions involving large numbers of participants many issues remain open.

A business process can be defined as a set of activities executed according to a defined set of rules in order to achieve a specific goal. When two or more organizations wish to embed long-running interactions within their business processes, the focus shifts from the inside of a process to interactions of this process with an external environment. What aspects of service interaction have to be explicitly modeled? How to classify a given interaction scenario? What standard supports a desirable interaction scenario, and which system to select for the realization of the cross-organizational interaction? Answering these questions is significant for understanding of the requirements for service interaction.

To specify requirements in service interaction more extensively than it is done in BPEL4WS and to assess emerging web standards, thirteen Service Interaction Patterns [9] covering bilateral, multilateral, competing, atomic and causally related interactions were identified. A systematic review of the thirteen Service Interaction Patterns presented in [9] has revealed that the scope of the patterns is limited to simple interaction scenarios and that they suffer from an ambiguous interpretation due to their imprecise definition. *In this paper*, we address these gaps by exploring additional possibilities for request-response interactions and by providing a precise formal semantics in the form of Coloured Petri Nets (CPNs) [10,11].

Instead of listing all patterns identified, we propose a framework that allows for a multitude of pattern variants to be generated by configuring a conceptual model of a generic service interaction scenario. The framework is built upon two concepts: a *pattern variant* and a *pattern family*. Every pattern family combines a set of pattern variants that are generated by assigning different values to every aspect of a generic service interaction scenario. Note that the original Service Interaction Patterns correspond to pattern variants belonging to different pattern families we identified. We also propose a notation which can be configured to represent different pattern variants graphically. The CPN models designed to formalize the semantics of the pattern family are also configurable and can be used to simulate the behavior of every pattern variant from the given pattern family.

We have identified five pattern families related to different aspects of message handling in the multi-party conversation (Multi-party Multi-message Request-Reply Conversation), publish-subscribe scenarios (Renewable subscriptions) and correlation on the low- and high-level of abstraction (Message Correlation, Bipartite Message Correlation and Tripartite Message Correlation) [12]. Due to the space limit, in this paper we describe only one pattern family Multi-party Multi-message Request-Reply Conversation. The pattern family presented can be used as a tool for evaluation of web services standards and tools. We implement some of the pattern variants from this family in Oracle BPEL PM and analyze the support of different pattern variants by WS-BPEL v2.0.

The remainder of the paper is organized as follows. Section 2 gives an overview of the related work. Section 3 presents the conceptual background and introduces the format for describing of pattern variants. The Multi-party Multi-message Request-Reply Conversation pattern family is described in Sec. 4. Section 5 shows the implementation of a selected pattern variant in Oracle BPEL PM. Evaluation of WS-BPEL v2.0 is performed in Sec. 6. This paper concludes with Conclusions described in Sec. 7.

2 Related Work

The Service Interaction Patterns documented by Barros et al. in [13,9] describe a collection of scenarios, where a number of parties, each with its own internal processes, need to interact with one another according to pre-agreed rules. These scenarios were consolidated into 13 patterns and classified based on the maximal number of parties involved in an exchange, the maximum number of exchanges between two parties involved in an interaction and whether the receiver of a response is necessarily the same as the sender of a request. Based on this classification four groups were identified: (1) single transmission bilateral interactions (i.e. one-way and round-trip bilateral interactions where a party sends and/or receives a message to another party); (2) single transmission multilateral non-routed interactions (i.e. a party sends/receives multiple messages to different parties); (3) multi transmission bilateral interaction (i.e. a party sends/ receives more than one message to/from the same party); (4) routed interactions.

Since the Service Interaction Patterns of Barros et al. [9] lacked a formal semantics, their formalization by means of the π-calculus has been proposed in [14]. Decker and Puhlmann formalized the patterns based on their descriptions, and did not take into account issues which were related to the patterns but which were incorporated into the pattern descriptions. Thus, they showed the possibility to formalize certain aspects of service interaction, but in fact did not make the definition of patterns less ambiguous. For example, the pattern Racing Incoming Messages specifies: *A party expects to receive one among a set of messages. These messages may be structurally different (i.e. different types) and may come from different categories of partners. The way a message is processed depends on its type and/or the category of partner from which it comes.* This pattern does not specify what happens if the party receives multiple messages at once, i.e. it is not clear how many of the received messages will be consumed and whether the rest of the messages will be discarded.

In [15] Zaha et al. formulate requirements for a service interaction modeling language, in addition to the ones covered by Barros et al. in [13]. The authors use these requirements for modeling behavioral dependencies between service interactions. In [16] Barros et al. introduce a set of correlation patterns that were used for evaluation of standards WS-addressing and BPEL. However, the framework presented by the authors does not cover relationships between different process instances. In [17] Barros et al. propose a compositional framework for service interaction patterns and interaction flows. They provide high-level models for eight service interaction scenarios using ASM, illustrating the need for distinguishing between different interpretations of the patterns.

In [18] Cooney et al. proposed a programming language for service interaction, which has been used to describe implementations of One-to-Many Send-Receive and Contingent Requests service interaction patterns [9].

Aldred et al. have performed a detailed analysis of the notion of (de-)coupling in communication middleware using three dimensions of decoupling, e.g. synchronization, time and space, and documented coupling integration patterns [19].

This work is also related to contracting workflows and protocol patterns of van Dijk [20], who proposed a number of protocol patterns for the negotiation phase on a transaction. Work of Hohpe and Woolf on Enterprise application integration [21] covers various messaging aspects that can be encountered during application integration.

Furthermore, this work relates to the Workflow Patterns initiative [22,23], where a set of 43 Control-flow patterns [24], a set of 40 Data patterns [25] and a set of 43 Resource patterns [26] are proposed. In addition, a Workflow Pattern Specification Language (WPSL) [27] has been defined which allows various pattern variants to be described in a language-independent way. In particular, the control-flow patterns have had a considerable influence on the development of new languages, the adaptation of the existing ones and all kinds of standardization efforts. This paper should be seen as a part of the Workflow Patterns initiative.

Our work, presented in this paper, differs from the described related work in the following aspects. We broaden up the scope of the original Service Interaction Patterns and systematically describe various pattern variants along with offering a graphical notation that is suitable for representing every pattern variant. To avoid ambiguous interpretation we formalize the patterns by means of CPNs.

3 Conceptual Background

In this section we describe concepts central to the pattern family considered and present the format for describing the pattern variants.

Instead of listing the whole set of patterns identified, we underline the differences between the pattern variants belonging to the same pattern family. We introduce the key concepts used in the pattern description by means of a UML Data Object diagram. The attributes that influence the detailed semantics of each pattern variant are described separately. To clarify the semantics of the pattern we apply the formalism of CPNs. We designed a (set of) CPN model(s) and tested them using the simulator facilities of CPN Tools. Declarations used within CPN models are based on the set of the concepts introduced in the UML diagram. We depict a generic service-interaction scenario belonging to a given pattern family with an icon graphically representing a set of attributes. By setting the attributes a specific variant of a pattern family is selected.

Pattern attributes (also referred to as parameters) represent the orthogonal dimensions for classifying different aspects of the service interaction within the context of the given pattern family. All possible combinations of the attribute values result in a large set of pattern variants, each of which can be easily derived from the generic service-interaction scenario and is depicted by a corresponding icon.

For the purposes of this paper a Conversation is defined as the communication of a set of contextually related messages between two or more parties. A Party is an entity involved in communication with other parties by means of sending/receiving messages. A party may represent a process, a service, a business unit, etc. A Message is a unit of information that may be composed of one or more data fields. A message may represent a request or a reply.

We describe the pattern family using the following format:

- *Description* of a generic pattern variant belonging to a given pattern family.
- *Examples* illustrating the application of the given pattern variant in practice.
- *UML meta-model* describing concepts specific to a given pattern family.
- *Visualization*: a graphical notation representing a generic pattern variant and the description of variation points that can be used for tuning the graphical notation to represent pattern variants.
- *CPN semantics*: the semantics of a generic pattern variant illustrated in the form of CPN models and their corresponding description.
- *Issues* that can be encountered when applying a pattern variant from the given pattern family in practice.

4 Pattern Family: Multi-party Multi-message Request-Reply Conversation

In this section, we present the Multi-party Multi-message Request-Reply Conversation pattern family using the format described earlier.

Description. A Requestor posts a compound request consisting of N sub-requests to a set of M parties and expects a reply message to be received for every sub-request. There exists the possibility that some parties will not respond at all and the possibility that a Responder will not reply on some sub-requests. The Requestor queues all incoming messages in a certain order. The enabling of the Requestor for consumption of reply messages depends on the fulfillment of activation criteria. The Requestor should be able to, optionally, consume a subset of the responses and even process a subset of the consumed set - hence allowing for business use cases where only the best or fastest responses are needed. The number of times the Requestor may consume messages from the queue can be specified explicitly.

Example

- A request to submit an abstract and to submit a paper is issued by an editor to a list of people registered for participation in a workshop. Only papers and abstracts submitted before a deadline would be reviewed. If a large amount of papers arrive, only the first 50 would be reviewed and only 10 best papers out of the reviewed ones would be published.

UML meta-model. An object diagram illustrating the pattern on the conceptual level is presented in Fig. 1. A Conversation consists of a set of messages (see a composition relation between `Conversation` and `Message`). A conversation involves an initiating process (e.g. Requestor), and at least one following process (e.g. Responder processes), depicted by associations `requestor` and `responder`. Any process may be or may not be involved in multiple conversations (see the multiplicity of the association `involves`). The Requestor generates at least one Request, while the Responder returns one or more Replies or does not react at all. The relation between request and reply messages is depicted by `corresponds to` association, and sending of request and reply messages by a party is illustrated by the dependency relations `is sent by` and `is produced by`. Requests issued by a Requester can be composite meaning that the Requestor may send several sub-requests in a single message concurrently to a single or multiple parties.

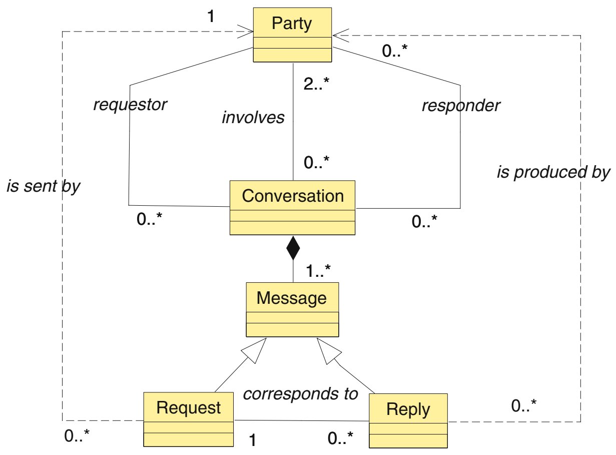


Fig. 1. UML meta-model of Multi-party Multi-message Request-Reply Conversation

Visualization. The graphical notation of the generic pattern variant is given in Fig. 2. The parties are visualized as rectangles. Directed arrows represent the direction in which a party sends a message. A message containing a single request is visualized as a black token, while a compound request is represented by multiple overlapping tokens. Parameters specific to a given party are visualized as icons residing within the boundaries of a rectangle representing a party. This graphical notation has the following set of *variation points*:

- N - a parameter denoting a list of sub-requests sent by a Requestor to a Responder in a single message.
  Range of values: size(N)$\geq$1.

Default value: size(N)=1.
Visualization: This parameter is depicted by the dots on the arc from Requester to Responder. If size(N)>1 or size(N)=1 the graphical notations depicted in Fig. 3 (1a) and (1b) are used respectively.

- *M* - a parameter denoting a list of Responders involved in the conversation.
Range of values: size(M)≥1.
Default value: size(M)=1.
Visualization: if size(M)>1 or size(M)=1 the graphical notations depicted in Fig. 3 (2a) and (2b) are used respectively.

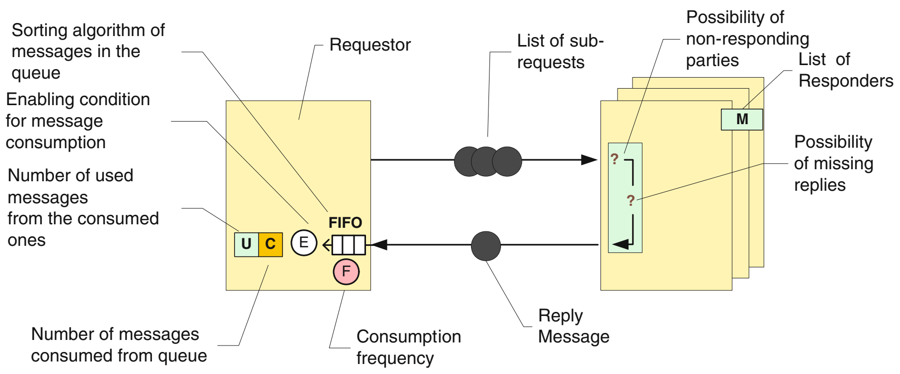

Sorting algorithm of messages in the queue
Requestor
List of sub-requests
Possibility of non-responding parties
List of Responders
Enabling condition for message consumption
M
Possibility of missing replies
Number of used messages from the consumed ones
FIFO
U C E F
Number of messages consumed from queue
Consumption frequency
Reply Message

Fig. 2. Graphical notation: Multi-party Multi-message Request-Reply Conversation

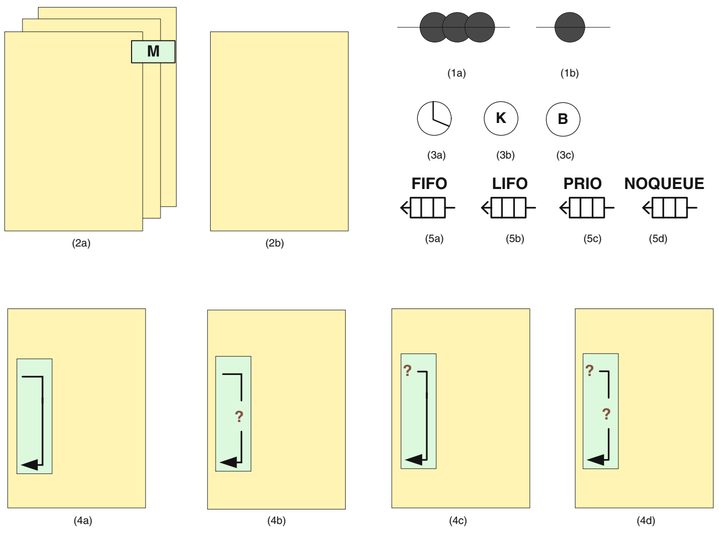


Fig. 3. Variants of graphical notation: Multi-party Multi-message Request-Reply Conversation

- *Possibility of non-responding parties* - a parameter specifying whether some of the Responders will ignore the request issued by the Requestor.
  Range of values:
  - No: all M Responders will reply at least something (for example, a request to report the level of income to the tax-office obliges all receivers to reply);
  - Yes: some Responders may not reply anything (for example, only interested parties react on the invitation to participate in a social event).

  Default value: No.
  Visualization: Fig. 3 depicts the graphical representation of four variations, where: in (4a) and(4b) all M Responders will produce at least some replies; in (4c) and(4d) some Responders may not reply on the requests received.
- *Possibility of missing replies* - a parameter specifying whether the Responder will not reply on some of the sub-requests (i.e. it is a choice of the Responder to engage in the conversation or not, and respectively to reply on all or only some of the received requests).
  Range of values:
  - No: Responders reply on all sub-requests (for example, the Responder answers on all questions in the tax declaration);
  - Yes: Responders reply only on some sub-requests (for example, a client subscribes only for two out of five journal offers received).

  Default value: No.
  Visualization: Fig. 3 depicts the graphical representation of four variations, where: in (4a) and (4c) no replies will be lost; in (4b) and (4d) some replies may not reach the Requestor.
- *Sorting of the queued messages* - a parameter specifying an ordering discipline according to which response messages queued by the sender are sorted.
  Range of values:
  - FIFO: oldest message is queued first;
  - LIFO: newest message is queued first;
  - PRIO: sorting based on some criterion (for instance, the price);
  - NOQUEUE: messages are not queued and consumed upon arrival if the sender is ready to process them, otherwise they are lost.

  Default value: FIFO.
  Visualization: Fig. 3 (5a)-(5d) depicts the graphical notation of different policies applied for sorting messages in the queue.
- *Enabling condition* - a parameter specifying the condition that has to be fulfilled to enable the Requestor to consume replies.
  Range of values:
  - a timeout (for example, requests for purchase on discount basis are accepted only until the expiration of the discount period);
  - a boolean condition, examining the properties of the queued messages (for example, at least three low-cost offers are required to select the best of them);

  - a specified number of messages K ($0<K\leq N$).

  Default value: K=1.
  Visualization: The icon *E* residing at the Requestor's side in Fig. 1 substituted with one of the graphical notations presented in Fig. 3 (3a), (3b) and (3c) which denote the enabling condition based on a timeout, availability of specific number of messages and boolean expression respectively.

- *Consumption index* - a parameter specifying the number of reply messages to be consumed by the Requestor from the queue.
  Range of values:
  - 0: none of the messages are removed from the queue (for example, messages must have enabled the process to receive, but it may need to leave them on the queue for another process to use);
  - S: S messages are removed from the queue such that $0\leq S<K$, where K is a number of replies sufficient for activation of the requester (for example, only messages selected by a boolean expression based on the property values are consumed);
  - All: all messages contained in the queue are removed.

  Default value: All.
  Visualization: The icon *C* residing at the Requestor's side in Fig. 2 substituted with a suitable value.

- *Utilization index* - a parameter specifying a number of messages from the consumed ones used by the Requestor for the processing.
  Range of values:
  - 0: no messages are used for processing (for example, if no messages were consumed, or if none of the consumed messages are required by the receiving process);
  - 1: one message is used for processing (for instance, a best offer from the available ones is selected);
  - UN: a number of messages used for the processing such that $1<UN<C$, where C is a number of messages consumed (for example, a boolean condition chooses only messages that pass the boolean constraint);
  - All: all consumed messages are used for the processing.

  Default value: All.
  Visualization: The icon *U* residing at the Requestor's side in Fig. 2 substituted with its value.

- *Consumption Frequency* - a parameter specifying the number of times the sender performs the consumption of messages from the queue.
  Range of values:
  - 1: the sender is activated only once, after this all remaining and arriving messages are destroyed;
  - FN: the sender consumes messages FN number of times, $1<FN$, after which all remaining and arriving messages are destroyed;
  - ∞: the sender consumes messages as long as they arrive.

Default value: 1.
Visualization: The icon F residing at the Requestor's side in Fig. 2 substituted with its value.

The pattern variant representing a scenario in which every parameter is set to the default value is presented in Fig. 4. A party A sends a single request to a party B, who sends a reply back. The party A queues the messages in the FIFO-order, and as soon as one message is received from the party B, it is consumed and processed. The presented notation may be used to represent Broadcast Remote Procedure Calls (RPC) [28], which expects one or more answers from each responding machine and treats all unsuccessful responses as garbage by filtering them out.

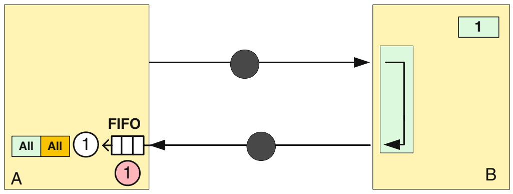


Fig. 4. Notation for the default pattern variant

CPN semantics. To avoid an ambiguous interpretation of the pattern variants related to Multi-party Multi-message Request-Reply Conversation we formalize the semantics by means of CPNs. Figure 5 depicts the top view of the CPN diagram representing the pattern.

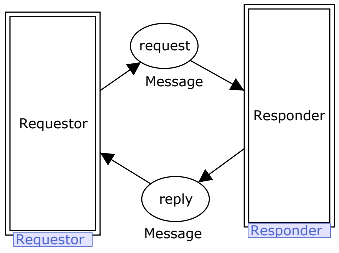


Fig. 5. CPN diagram: The main view

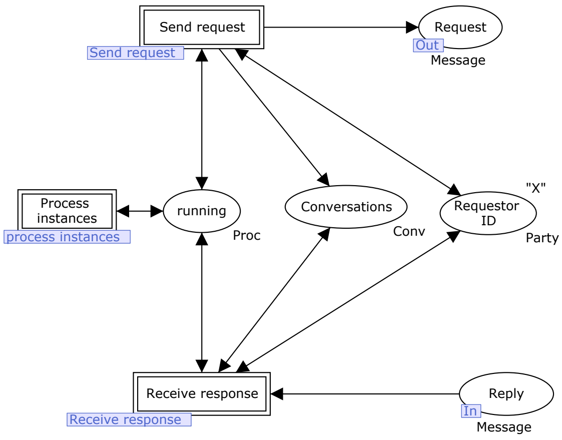


Fig. 6. CPN diagram: The `Requestor` page

Requestor and Responder are represented as substitution transitions which can be unfolded to the nets depicted in Fig. 6 and Fig. 7(c) respectively. In every given conversation the parties exchange requests and replies of type `Message`.

The Requestor (whose behavior is shown in Fig. 6) can send requests and receive response messages using substitution transitions `Send request` and `Receive response` whose decomposition is presented in Fig. 7(b) and Fig. 8. A Requestor process may have multiple process instances, whose lifecycle is shown in Fig. 7(a). Process instances available for participation in a conversation are stored in place `enabled`. When for a given process instance a conversation is started, a conversation identifier `cid` is coupled with a process instance. The uniqueness of identifiers is ensured by incrementing of a counter whose value is stored in place `Conversation counter`. A process instance chosen for conversation is stored in place `running`. Transitions `activate`, `deactivate` and

Table 1. Data types used in Figs. 5-8

```
colset Party = string;
colset Request = string;
colset Requests = list Request;
colset Reply = string;
colset Replies = list Reply;
colset ConvId = int;
colset Content = union Req:Requests + Repl:Replies; [a]
colset Message = product ConvId * Party * Party * Content; [b]
colset Count = int;
colset MTime = int;
colset Status = with active|inactive|enabled|completed; [c]
colset ConvRequest = product Parties * Requests;
colset ConvRequests = list ConvRequest;
colset ConvReply = product Parties * Replies;
colset ConvReplies = list ConvReply;
colset Pr = product ConvRequests*ConvReplies*Status;
colset Proc = product ConvId*Pr; [d]
colset ConvInfo = record start_time: MTime * last_act:MTime
    * nof_unique_messages: Count * nof_parties: Count * total_nof_messages: Count;
colset Conv = product ConvId * ConvInfo * Status;
```

[a] The content of a message is either a list of requests or a list of replies. The CPN union type is used to specify this.

[b] A message is a tuple `(cid,P1,P2,c)` where `cid` is a conversation identifier, `P1` is the requestor, `P2` is the responder, and `c` is the content. Such a message is of type `Message`.

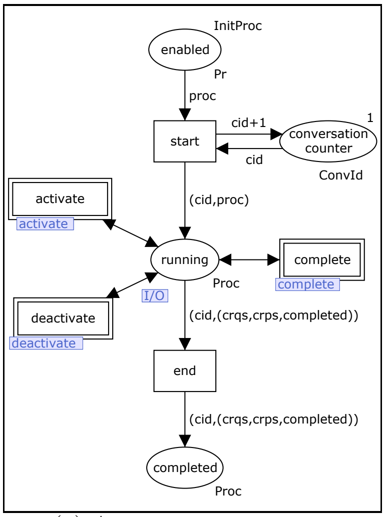
[c] The lifecycle of a process instance starts with activation of an `enabled` instance. An `active` instance can become `inactive` through deactivation, or `completed` when the instance lifecycle ends.

[d] Process instances of type `Pr` contain a list of requests sent, replies received and a status of the instance. When a conversation starts, a process instance is coupled with a conversation identifier.

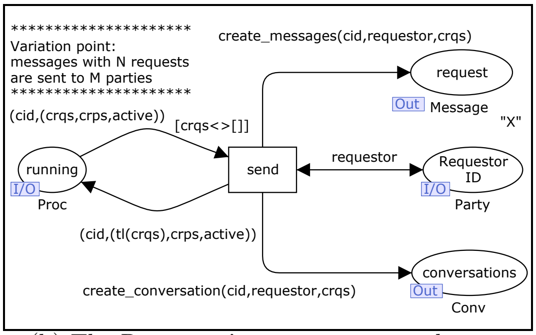
complete control the status of a process instance during its lifecycle. When an enabled process instance is activated, it gets the status active and may participate in sending and receiving of messages. Meanwhile the active process instance can become inactive through deactivation or can become completed. The lifecycle of a process instances ends upon completion and the process instance is placed to place completed.

The Requestor's Send Request sub-page in Fig. 7(b) shows that the Requestor, whose identifier is stored in place Requestor ID, on the moment of sending a request message creates a new conversation. Function create_messages() takes a list of conversation requests crqs of type ConvRequests, which contains a list of parties to whom a request should be sent, and a list of sub-requests that should be sent to each party, and creates as many messages as there are parties in the list. This function directly corresponds to the variation point specifying that messages with N sub-requests are sent to M parties.

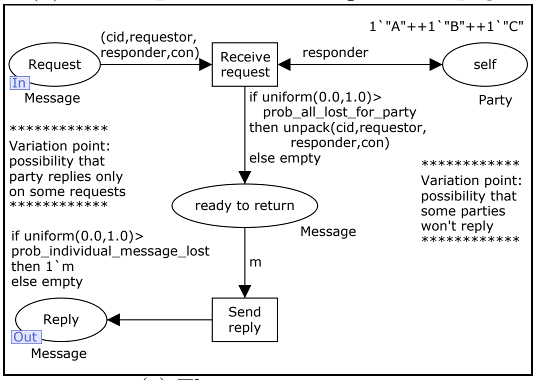
When request messages are created, a new conversation is created by means of the function create_conversation(). This function records the information about the conversation identifier, conversation-specific parameters (the start time of the conversation, the time of the last activation, a total number of



(a) A Process instances sub-page



(b) The Requestor's Send Request sub-page



(c) The Responder page

Fig. 7. CPN diagrams

messages sent, a number of parties to whom the requests have been sent, and a number of unique messages (i.e. a number of sub-requests contained in the single message)), and the status of the process instance. The recorded conversation information is used later on for the purpose of correlating response messages received with the requests sent and for identifying how many times the received messages can be consumed for processing.

The Responder page shown in Fig. 7(c) illustrates the behavior of Responders involved in the conversation. The identifiers of the Responders are stored in place `self`. They are used to relate incoming requests to a right party, based on the party identifier. When a Responder receives a request message, it unpacks the composite requests into separate messages each containing a separate sub-request. The parameter `prob_all_lost_for_party` corresponds to a variation point specifying the probability that the Responder will ignore a received composite request or will process it. If the Responder decides to reply on the request, the parameter `prob_individual_message_lost` is used as a variation point to define the probability that a reply will be sent for every unpacked sub-request.

The Requestor's `Receive response` sub-page presented in Fig. 8 illustrates the mechanism of queueing and processing of incoming responds by the Requestor. The Requestor processes only messages addressed to it (for this purpose, a `Requestor ID` is used). The response messages received are queued according to the `QueueingDiscipline()` function, which corresponds to a variation point that can be set to any of the queueing disciplines, i.e. LIFO, FIFO or PRIO.

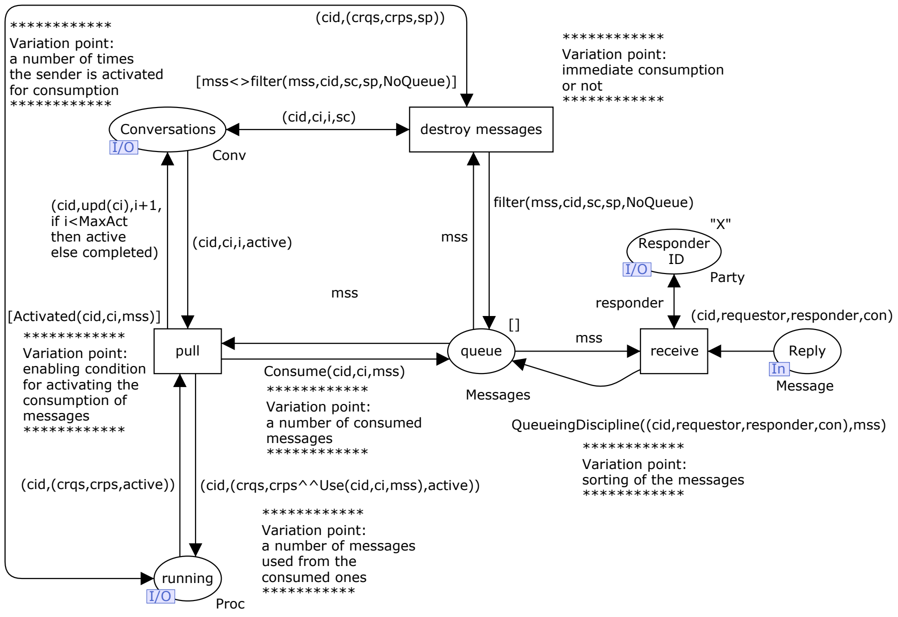


Fig. 8. CPN diagram: The Requestor's `Receive Response` sub-page

Function `Consume()` corresponds to a variation point specifying how many messages from the queued ones have to be consumed. One, several, or all available in the queue messages can be consumed. The consumption of messages happens upon the satisfaction of an enabling condition (which is a variation point) encoded as a guard of transition `Pull`. Function `Activated()` can be tuned to specify the enabling upon the availability of one or several messages in a queue, upon the satisfaction of a certain condition, or upon a timeout.

From the messages consumed only a number of messages defined by the function `Use()` are actually used by the Requestor for the processing. This variation point can be set for using either one, several or all consumed messages.

The parameter `MaxAct` corresponds to the variation point specifying how many times the Requestor may consume the messages from the queue for the given conversation. If the messages have been consumed the specified number of times, the process instance receives the status `completed` and the messages left in the queue are removed from it by means of the function `filter()`. Transition `destroy messages` is used to retrieve messages from the place `queue` if the incoming response messages do not need to be sorted and have to be consumed immediately upon arrival.

Issues. When applying pattern variants belonging to the Multi-party Multi-message Request-Reply Conversation pattern family an issue of the message correlation may arise while matching replies received with the requests sent. This issue can be solved by applying a suitable pattern variant from the Message Correlation family. If a Multi-message Multi-Party Request-Reply Conversation pattern variant has to be applied in the context of a long-running conversation, where a series of requests have to be sent one after another, the given pattern variant can be combined with a suitable pattern variant from the family of Bipartite Conversation Correlation.

5 Oracle BPEL PM: A Default Scenario in Action

In this section, we illustrate an implementation of the default pattern variant in Oracle BPEL PM v.10.1.3.1.0 (which is a tool supporting design of BPEL processes).

Figure 9(a) illustrates an asynchronous process which upon an initiation by a client performs the invocation of a synchronous service `ResponseProcess` presented in Fig. 9(b) using an invoke activity `SendRequestToResponder`.

```
<invoke name="SendRequestToResponder" partnerLink="ResponseProcess"
        portType="ns2:ResponseProcess" operation="process"
        inputVariable="RequestorInputVariable"
        outputVariable="ObtainedOutputVariable"/>
```

The content of the request sent, enclosed in the `RequestorInputVariable`, is specified by the <assign> activity `AssignInputData` in Fig. 9(a). The Responder process `ResponseProcess` is initiated by a message received from the

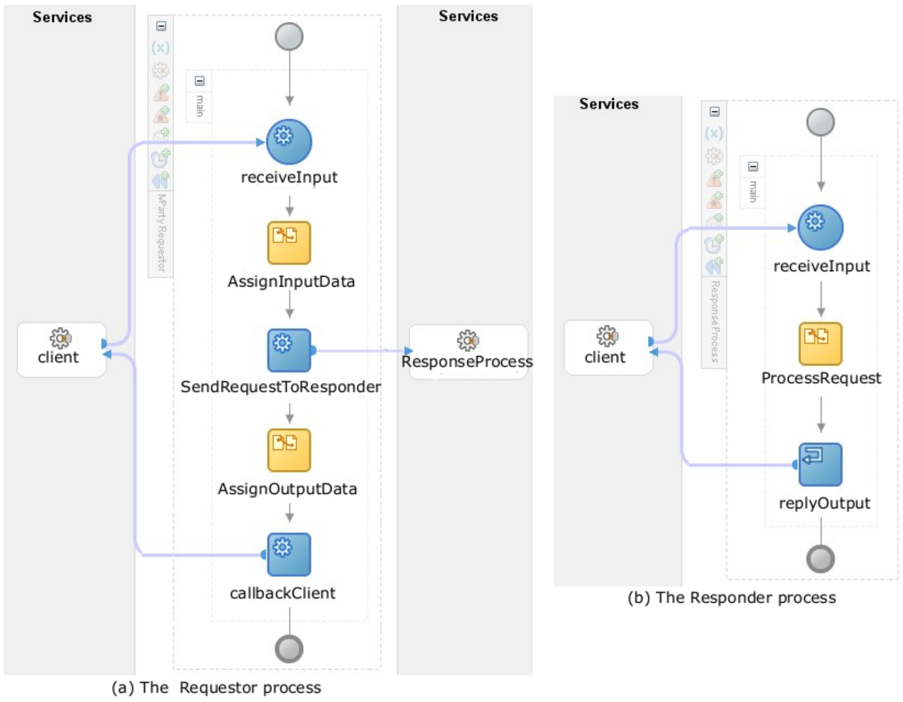


Fig. 9. Implementation of default pattern variant in Oracle BPEL PM

Requestor process. The request received is processed by an <assign> activity `ProcessRequest` in Fig. 9(b) and a response is sent back to the Requestor process using a `replyOutput` activity. The response message is assigned to an output variable `ObtainedOutputVariable` of the `SendRequestToResponder` invoke activity. Note that <invoke> activity has no attribute for message queueing, therefore response messages are not queued and are consumed and processed as soon as they arrive. We discuss the support of other pattern variants by WS-BPEL v.2.0 in the next section.

6 Evaluation of WS-BPEL v2.0

In this section we analyze what pattern variants of Multi-party Multi-Message Request-Reply Conversation are supported by WS-BPEL v2.0 by defining what values each variation point can take.

- *Number of sub-requests in a message*: an `<invoke>` activity in WS-BPEL is used to call an operation on a service. To realize a request-reply conversation a correlation pattern of the `<invoke>` activity has to be set to "request-response" and both an `inputVariable` and `outputVariable` of certain data types have to be specified. Since WS-BPEL is XML-based, a complex data

types can be defined, thus allowing to compose message from multiple sub-requests.
- *Number of Responders involved in a conversation*: many parties can be involved in a conversation with a given Requestor. A message can be sent by a Requestor to a set of Responders in parallel if every Responder is defined as a separate `PartnerLink` and a separate `<invoke>` activity is placed for every partner either in the body of the `<flow>` construct or in the `<forEach>` construct that operates in parallel mode. Therefore WS-BPEL can implement this, albeit clumsily.
- *Possibility of non-responding parties*: an `<invoke>` activity is used to call (an operation on) a service. Such an invocation can be one-way or request-response. When a request-response invocation is performed by the Requester, the `<invoke>` activity stays open until the response is received. This however does not guarantee that the service invoked will respond.
- *Possibility of missing replies*: in WS-BPEL inbound message activities (IMA) (i.e. `<receive>`, `<pick>`, `<onEvent>`) may complete only after they have received a matching message. However, in some situation an *orphaned IMA* occurs when an inbound message activity remains to be open. In this case, the standard fault `bpel:missingReply` is thrown and the orphaned IMA is not considered to be orphaned anymore.
- *Sorting of queued messages*: messages received by a process instance are not queued (NOQUEUE). WS-BPEL defines that a receiving activity needs at most one message to proceed. However, in the situation when a receiving activity is not ready for consumption and multiple messages arrive at a time, a race condition occurs. WS-BPEL does not mandate any specific mechanism for handling race conditions and leaves this decision to the BPEL engine designers.
- *Enabling condition*: an inbound message activity becomes enabled as soon as a matching message has been received by a process instance (i.e. a message of a specific type). However, it is also possible to use a `<wait>` construct in order to enable an activity for message receival after a given time period or after a certain deadline has been reached.
- *Consumption index*: only one message at a time can be consumed by an inbound message activity.
- *Utilization index*: since inbound message activities can consume only one message at at a time, therefore the message consumed is also the one used for processing.
- *Consumption Frequency*: in WS-BPEL it is possible to specify that a party may consume messages multiple times if IMA is placed in a `<while>` or `<repeatUntil>`. The consumption frequency in this case is defined by the evaluation of the boolean condition defined in these repetitive constructs.

The mapping of the pattern attributes to the WS-BPEL is not straightforward, since WS-BPEL does not have concepts able to capture the meaning of all pattern attributes or these concepts are not explicitly defined. By definition, all inbound message activities in WS-BPEL are executed as soon as a suitable message arrives. Selection of such a behavior as a default results in quite limited capabilities of WS-BPEL to support different variants of message handling. Since WS-BPEL intentionally does not specify a mechanism for handling of the race conditions, systems supporting BPEL-processes may employ different implementations and thus support distinct pattern variants. In this case, the pattern attributes can be used to assist in selecting an appropriate system.

7 Conclusions

The approach presented in this paper shows that a multitude of pattern variants can be derived by assigning different values to variation points identified as a result of the systematic analysis of service interaction scenarios. This approach is applicable for describing other problems in the form of the configurable framework, given that all dimensions of the problem analyzed are clearly delineated and well understood. The main benefit of presenting patterns by means of a configurable pattern family is that it allows various variants of multi-dimensional complex problems to be described and referenced in a uniform way. The complexity of the Multi-party Multi-message Request-Reply Conversation pattern family is characterized by 6912 pattern variants (this number is calculated as multiplication of total number of values each of the variation points may take). The variation points identified can be used for the evaluation of tools and web-service composition standards as it has been done for Oracle BPEL PM and WS-BPEL v2.0. The analysis of WS-BPEL has shown that many pattern variants related to processing of multiple messages are not supported. Such an analysis forces us to deeply think about the requirements in service interaction and may trigger a revision of current best practices in order to capture all variation points and support more pattern variants. Furthermore, the pattern family presented can be used as a solution selection instrument, or even as a set of requirements for new languages in the area. In the future, we plan to use the pattern families as a benchmark for classification of complex service interaction scenarios.

References

1. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H., Thatte, S., Winer, D.: Simple Object Access Protocol (SOAP) 1.1. (2000), http://www.w3.org/TR/soap
2. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1 (2001), http://www.w3.org/TR/wsdl
3. Belwood, T., et al.: UDDI Version 3.0 (2000), http://uddi.org
4. Arkin, A., Askary, S., Fordin, S., Jekel, et al.: Web Service Choreography Interface (WSCI) 1.0. Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems (2002)
5. Arkin, A., et al.: Business Process Modeling Language (BPML), Version 1.0 (2002)
6. Thatte, S.: XLANG Web Services for Business Process Design (2001)
7. Peltz, C.: Web services orchestration: a review of emerging technologies, tools and standards. Hewlett Packard, Co. (2003)

8. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation (2003)
9. Barros, A., Dumas, M., Hofstede, A.: Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection. QUT Technical report, FIT-TR-2005-02, Queensland University of Technology, Brisbane (2005)
10. CPN Group University of Aarhus, Denmark: CPN Tools Home Page http://wiki.daimi.au.dk/cpntools/
11. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. In: EATCS monographs on Theoretical Computer Science, Springer, Berlin (1992)
12. Mulyar, N., Aldred, L., Aalst, W., Russell, N.: Service interaction patterns: A configurable framework. BPM Center Report BPM-07-07, BPM Center, BPMcenter.org (2007)
13. Barros, A., Dumas, M., ter Hofstede, A.: Service Interaction Patterns. In: Proceedings of the 3rd International Conference on Business Process Management, Nancy, France, vol. 3716/2005, pp. 302–318 (2005)
14. Decker, G., Puhlmann, F., Weske, M.: Formalizing service interactions. In: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, pp. 414–419. Springer, Heidelberg (2006)
15. Zaha, J., Barros, A., Dumas, M., ter Hofstede, A.: Let's Dance: A Language for Service Behavior Modeling. In: OTM Conferences (1), Vienna, Austria, pp. 145–162 (2006)
16. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation Patterns in Service-Oriented Architectures. In: FASE. Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering, Braga, Portugal (2007)
17. Barros, A., Borger, E.: A Compositional Framework for Service Interaction Patterns and Interaction Flows. In: Lau, K.K., Banach, R. (eds.) ICFEM 2005. LNCS, vol. 3785, pp. 5–35. Springer, Heidelberg (2005)
18. Cooney, D., Dumas, M., Roe, P.: GPSL: A Programming Language for Service Implementation. In: Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering, Vienna, Austria (2006)
19. Aldred, L., Aalst, W., Dumas, M., Hofstede, A.: Understanding the Challenges in Getting Together: The Semantics of Decoupling in Middleware. BPM Center Report BPM-06-19, BPMcenter.org (2006)
20. van Dijk, A.: Contracting Workflows and Protocol Patterns. In: Business Process Management, pp. 152–167. Springer, Heidelberg (2003)
21. Hohpe, G., Woolf, B.: Enterprise Integration Patterns. Addison-Wesley Professional, Reading (2003)
22. Aalst, W., Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
23. WPHP: Workflow Patterns Home Page, http://www.workflowpatterns.com
24. Russell, N., Hofstede, A., Aalst, W., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, BPMcenter.org (2006)
25. Russell, N., Hofstede, A., Edmond, D., Aalst, W.: Workflow Data Patterns. QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane (2004)

26. Russell, N., Hofstede, A., Edmond, D., Aalst, W.: Workflow Resource Patterns. In: WP 127, Eindhoven University of Technology, Eindhoven. BETA Working Paper Series (2004)
27. Mulyar, N., Aalst, W., ter Hofstede, A.H.M., Russell, N.: Towards a WPSL: A Critical Analysis of the 20 Classical Workflow Control-flow Patterns. Technical report, Center Report BPM-06-18, BPMcenter.org (2006)
28. Broadcast RPC: Programming with Remote Procedure Calls. http://ou800doc.caldera.com/en/SDK_netapi/rpcpC.bcast.html

Building Adaptive Systems with Service Composition Frameworks

Liliana Rosa, Luís Rodrigues, and Antónia Lopes

Faculty of Sciences, University of Lisbon, Portugal
lrosa@lasige.di.fc.ul.pt, {ler,mal}@di.fc.ul.pt

Abstract. Frameworks that support the implementation and execution of service compositions are a fundamental component of middleware infrastructures that support the design of adaptive systems. This paper discusses the requirements imposed by adaptive middleware on service composition frameworks, and discusses how they have been addressed by previous work. As a result, it describes the design of a novel adaptation-friendly service composition framework that takes into consideration the requirements at three different levels: service programming model level, adaptation-friendly services level, and kernel mechanisms level.

1 Introduction

Today's applications need to be designed to operate in a wide range of heterogeneous devices, including servers, PCs, PDAs, or mobile phones. Given this diversity, it is fundamental to be able to design and deploy adaptive applications. An adaptive application is able to change its behavior to better match the (functional and non-functional) expectations of the user. For instance, by adjusting the multimedia quality exchanged among different participants, according to the available network bandwidth.

Unfortunately, building distributed applications that can monitor changes in their execution environment, as well as in the user requirements, and react to those changes by adapting their behavior is an inherently complex task. A task that can be greatly simplified by the use of appropriate adaptive middleware. A key component of a middleware platform to support the construction and execution of adaptive applications is a software framework that facilitates the composition of services. By allowing services to be composed in different manners, and supporting the dynamic reconfiguration of service compositions, it becomes easier to adapt the behavior of applications that are built in a modular manner.

Network protocols have been specified for a long time in a modular way, using the layer abstraction. Typically, a communication system is built from a vertical composition of multiple protocols layers. Therefore, it comes as no surprise that many software frameworks to build configurable communication services have been designed, implemented, and used in different contexts. Some of the most relevant protocol composition frameworks are x-kernel [1], *Cactus* [2], *Horus* [3], *Ensemble* [4], *Appia* [5], *Eva* [6], and *Samoa* [7]. Given the significant amount of experience that has been gathered with these systems, they become obvious candidates to inspire the construction of a service composition framework to include as part of a middleware platform to support adaptive applications.

This paper looks at existing protocol composition frameworks from the point of view of their adequacy to support the implementation of adaptive services. Based on our experience in building a generic architecture to support adaptation [8], we identify a number of requirements that need to be satisfied by any service composition framework. We then analyze how existing protocol composition frameworks address these requirements. The contribution of this paper is the identification of a set of features, lacked by many of the existing protocol composition frameworks, that are key to support the dynamic reconfiguration of service compositions. Moreover, we describe how we have addressed these requirements in the implementation of an adaptation-friendly service composition framework named *RAppia*.

The rest of the paper is structured as follows. Section 2 introduces protocol composition frameworks. Section 3 identifies a set of requirements imposed by adaptive middleware on composition frameworks, and Section 4 analyses how these are addressed by existing frameworks. The design and implementation of *RAppia* is described in Section 5. Finally, Section 6 concludes the paper.

2 Protocol Composition Frameworks

Protocol composition and execution frameworks aim at simplifying the design, implementation, and configuration of communication protocols. One of the main goals of such frameworks is to promote the design of communication services in a modular way, by encouraging communication functionality fragmentation in different modules, that can be composed in several ways. As a result, the designer has the opportunity to compose communication services that exactly match the application needs. A second important goal of these frameworks is to provide an efficient execution environment for protocol compositions, by providing runtime services that support the exchange of data and control information among components, time management services, buffer management services, etc.

The reader should be aware that there are similarities among composition frameworks and general purpose operation systems. Typically, an operation system includes a kernel, services that can be implemented partially in the kernel and partially in user level (such as windows management), a number of user level services (for instance, the command interpreter and system utilities), and a programming model (processes, file system interface, synchronization primitives, etc). In some sense, a service composition framework is a specialized operating system (in fact, one of the first protocol composition services was even called a "kernel" [1]). Thus, in this paper, when we refer to service composition frameworks we analyze them taking a global perspective, considering both the kernel functionality, the services typically provided with the framework, and the programming model enforced by it.

Multiple protocol composition frameworks have been built [1,2,3,4,5,6,7]. Although all of these frameworks aim at achieving similar goals and are based on the same foundations, inspired by the original work of *x-kernel* [1], there are some significant differences among them.

In *x-kernel*, *Horus*, *Ensemble*, *Cactus*, and *Appia*, communication among protocols is performed by the exchange of events. With the exception of *Cactus*, all frameworks support vertical protocol compositions, i.e., events are processed in order by all the protocols in the stack. In *Cactus*, events can be processed in parallel. In *x-kernel* and *Horus* events are delivered to all protocols (which may process them or just forwards them in the stack). *Ensemble* proposed some offline tools to extract fast-paths for most common events. *Appia* and *Cactus* allow each protocol to subscribe only the events it is interested in processing. On the other hand, both *Samoa* framework [9] and *Bast* [10] protocol library follow different approaches. *Samoa* also relies in protocol compositions but with a service-based design. Therefore, the framework kernel is particularly different from the remaining frameworks. In this case, the interaction between protocols is achieved using remote method invocations. In this approach, each module exports a set of executers, listeners, and interceptors, each being responsible for a different service: requests, replies, and notifications. In *Bast* each protocol is an object, thus, interaction relies in method invocation, and the composition model is not strictly vertical.

Among these frameworks, only *Cactus* [11], *Ensemble* [12], and *Samoa* [9] have addressed the problem of dynamic adaptation, supporting the runtime reconfiguration of protocol compositions. However, these efforts have considered only a limited set of protocols (for instance, group communication, in the case of *Ensemble*) and specific reconfiguration strategies. As we will discuss later in the text, none of these frameworks can claim to provide generic support for multiple reconfiguration strategies.

3 Adaptation Requirements

Our previous work on the development of a generic architecture to support the adaptation of service compositions [8,13], gave us insight on the needs, challenges, and goals that adaptation brings. The highlights of adaptation are related to context monitoring, to detect changes that will trigger adaptation, and adaptation management, that conducts and performs all the process of reconfiguring the composition. Our experience allowed to identify several requirements that have to be satisfied by composition frameworks. We note that different requirements impact different aspects of the composition framework: some require changes to the runtime support provided by the framework (also known as the framework kernel), some can be satisfied by adding additional services to the framework, others affect the programming model enforced by the framework. These requirements are identified and described in detail in the following sections.

3.1 Context Monitoring

The context information characterizes the execution context. Since the execution context may change with time, this information has a dynamic nature. Thus, given that the execution context is dynamic, a particular configuration of the application, that was adequate in given context may become inadequate later on, and require adaptation. Therefore, it is of utmost importance to maintain the context under constant monitoring, such that the context information reflects the current state of the environment.

The context information may include information from different sources, ranging from user preferences to hardware characteristics of devices hosting the application [14]. This information can be generated by the services themselves, or captured from other origins, such as the operating system or the device. The information itself can be used to infer other context properties, i.e. higher level context information, such as system stability, based on low-level context information such as network error rate, connectivity information, etc. Context information capture can be performed on-demand, or continuously, in a periodic manner [15]. Moreover, services can produce notifications that signal infrequent occurrences, such as the failure of a component, or that some control variable exceeded a predefined threshold. From these observations on context capture, the following requirements can be identified:

Requirement 1: The composition framework should support a programming model that makes easier for sources of context information to make this information easily accessible (in particular when these sources are the composable service implementations themselves).

Requirement 2: The composition framework should provide the mechanisms to support the capture of context information, both continuously or on-demand, as well as mechanism to handle notifications generated by context sources.

To perform adaptation is not enough to gather context information; it is also necessary to analyze the collected information in order to detect relevant changes. The analysis can be directly embedded in the mechanisms used to collect the context information or may be performed by an external component. In either way, the following requirement can be identified:

Requirement 3: The composition framework should include, or be augmented with, services that are able to analyze the context information and report relevant changes.

3.2 Reconfiguration Actions

In this paper, we are concerned with the construction of adaptive distributed systems whose adaptation logic can be separated form the core application logic. In this way, it is assumed that the structure of the application is organized into two discrete layers, with the core application logic built on the top of a composition of domain-specific and general middleware services. Adaptiveness results from the dynamic reconfiguration of this composition of services, in reaction to changes in the users' preferences or in the execution context.

There are two main ways in which the application may be adapted. To start with, the behavior of each individual service may be adapted, usually by setting predefined configuration parameters [16,17]. Furthermore, when an appropriate composition framework is used, one may also change the services included in the service composition and the way these services are composed [18,19]. When we restrict ourselves to communication services, reconfiguration of the composition boils down to the addition, removal, or exchange of protocols. Therefore, we identify the following requirement:

Requirement 4: The composition framework has to provide support for dynamic reconfiguration, including mechanisms to perform parameter configuration, and mechanisms to perform the addition, removal, and exchange of services to a given composition.

When applying a reconfiguration action, the correctness of the service composition has to be preserved. To achieve this goal, several issues need to be addressed during the reconfiguration process. A first issue is related to the amount of required synchronization among the nodes involved in the reconfiguration. For instance, in some cases, each node may perform the local reconfiguration of the service composition without explicit coordination with other nodes; in other cases, a node may not be allowed to proceed with the local reconfiguration until it becomes aware that all the other nodes are also ready to reconfigure. Another issue is related to the state information that may have to be transferred from one system configuration to the other. The third issue is related to the dynamics of each individual service during reconfiguration. Namely, in some cases, a service may be required to be placed in a quiescent state before reconfiguration is performed. Note that different services impose different constraints on the way issues above are handled and, for any given service, different reconfiguration actions may also impose different constraints [13]. Thus, the mechanisms enumerated should be rich enough to satisfy a wide range of constraints, such that the reconfiguration may be performed with the minimal interference on the execution of the services in the composition. This results in the following requirement:

Requirement 5: The composition framework should provide, either embedded in its kernel or as a set of additional services, a comprehensive set of mechanisms to support the coordination among nodes, to transfer service state information between services, and to enforce a quiescent state of a service.

3.3 Selection of Adaptation Targets

We are interested in building *distributed* adaptive applications. Therefore, service compositions will be executed in multiple nodes of the system. As a result, when a reconfiguration needs to be performed, it may need to affect all nodes or just a subset of the nodes involved in the application. Furthermore, only a subset of the service composition may be affected by the reconfiguration.

When specifying the adaptation logic of a system, it is very hard to specify it in a generic and reusable manner if one is required to explicitly name each individual instance of every service that is affected by the adaptation. On the contrary, it is much more powerful to specify the adaptation target indirectly, for instance, using service type hierarchies or using meta-information [20] to tag all services with their properties. The service composition framework may contribute to simplify the implementation of an adaptive system if it provides the programming abstractions and the runtime mechanisms that allow to map these high level abstractions (such as service type hierarchies) in run-time artifacts, for instance, using a reflective approach. Thus:

> **Requirement 6:** The composition framework should provide mechanisms to reason or obtain information on the system.

4 Adaptation Support in Existing Composition Frameworks

To understand the suitability of protocol composition frameworks for adaptation, it is important to analyze how each of the requirements identified in the previous section already is, or can be satisfied, by existing protocol composition frameworks.

4.1 Addressing the Requirements

> **Requirement 1:** The composition framework should support a programming model that makes easier for sources of context information, in particular when these sources are the composable service implementations themselves, to make this information easily accessible.

Most composition frameworks that have been developed to support protocol composition are event-based, i.e., different services communicate by exchanging events. Thus, the preferable method to make context information available is via the exchange of context events. The event model simplifies the implementation of context notifications: a service that wants to provide a notification about a relevant change in the context information needs simply to create and trigger a new *ContextNotification* event. When context information needs to be read on demand, each service must be ready to process *ContextQuery* events and respond with *ContextAnswer* events.

At first sight, it may seem that every protocol composition framework is equally fitted to satisfy this requirement. However, there are a number of implementation and modelling issues that have a significant impact on how this support is provided. To start with, context information is often service specific. Thus, the programmer will likely need to refine the base events provided by the framework. Thus, the composition framework cannot limit the set of events exchanged among services to a set of fixed events defined a priori (as, for instance, the *Horus* system). Furthermore, when context is read on-demand, a *ContextQuery* event needs to be delivered to all services that can potentially answer the query. To avoid the event to be delivered to every service of the composition and avoid a performance overhead, the framework should allow each service to explicitly list which events it is interested in (to our knowledge, only *Cactus* and *Appia* support this feature). Finally, the framework should encourage programmers to proactively provide support for context gathering in the service implementations. Thus, the events such as *ContextNotification*, *ContextQuery*, and *ContextAnswer* should make part of the service implementation model. To our knowledge, none of the existing protocol composition frameworks provides this feature explicitly.

Requirement 2: The composition framework should provide the mechanisms to support the capture of context information, both continuously or on-demand, as well as mechanism to handle notifications generated by context sources.

When building distributed adaptive applications the adaptation policy typically depends on the global context, i.e., of the aggregate context information collected from the different participants in the system. Therefore, it is not enough to support the local gathering of information. Each node should provide support for exporting context information to other nodes. To support on-demand reading of context information, each node must accept remote invocation from other nodes. To disseminate context information, nodes should be connected to a context dissemination bus. This type of support can be added to any of the existing composition frameworks, given that it may be implemented as a set of additional services. Still, to our knowledge, no composition framework includes such services in their distributions, although a fairly detailed pattern language [21] could be used to provide the necessary support.

Requirement 3: The composition framework should include, or be augmented with, services that are able to analyze the context information and report relevant changes.

As soon as it is possible to gather and distribute local context information, it becomes possible to analyze and interpret this information to extract the relevant information for the adaptation. Although the analysis can be potentially executed in a single central location, that collects all the context information gathered from all the nodes in the system, in some cases this approach may introduce inefficiencies in the system. For instance, consider that, for adaptation purposes, one is concerned with the average value of a context variable measured in a specific node in the system. The average could be computed at a central location, based on multiple remote readings of the context variable. However, it is possible to save signaling traffic, if the average is computed directly at the source node of the context information. To support the later approach, it is required that the context gathering and dissemination subsystem can be built as a composition of services itself. This is possible to achieve with any of the existing protocol composition frameworks.

> **Requirement 4:** The composition framework has to provide support for dynamic reconfiguration, including mechanisms to perform parameter configuration, and mechanisms to perform the addition, removal, and exchange of services to a given composition.

Although all existing protocol composition frameworks support offline configuration of the service compositions, only a few support the modification of the composition in runtime. From those that support dynamic reconfiguration, some severely restrict the way a composition may be reconfigured in runtime. For instance, *Ensemble* only supports the replacement of a vertical composition (a protocol stack) to another (even when both stacks have several layers in common), avoiding the problems caused by having part of the composition operational while the rest is being changed. From this point of view, *Cactus* is the most flexible of all existing composition framework, as it allows for services to be added and removed in runtime without restrictions.

The reconfiguration process can be also simplified if the addition, removal, and exchange of services to a given composition can be controlled from a remote node (for instance, a reconfiguration manager). This means that the composition framework should include a monitor able to interpret reconfiguration commands that may be activated, for instance, via remote invocations. To our knowledge, none of the existing frameworks supports such interpreter.

> **Requirement 5:** The composition framework should provide, either embedded in its kernel or as a set of additional services, a comprehensive set of mechanisms to support the coordination among nodes, to transfer service state information between services, and to enforce a quiescent state of a service.

Several protocol composition frameworks, such as *Ensemble*, *Cactus*, or *Samoa*, have implemented concrete instances of the mechanisms enumerated above. However, these mechanisms are usually designed with the goal of implementing a small number of predefined reconfiguration strategies, i.e, a particular sequence of operations such as coordination, enforce quiescent state, state transfer, etc. For instance, *Ensemble* implements a reconfiguration strategy that requires the composition of each node to reach a quiescent state; the state is then captured; a new composition is instantiated and the state loaded into the configuration at every node; finally, the new composition is restarted. *Cactus* and *Samoa* offer more efficient strategies but, in practice, the mechanisms supported only serve the predefined, built-in, strategies, and are only applicable in a limited number of situations. To our knowledge, no composition framework as attempted to offer a library of mechanisms required to support the coordination among nodes, to transfer service state information between services, and to enforce a quiescent state of a service that can be combined in different manners to implement multiple strategies.

Requirement 6: The composition framework should provide mechanisms to reason or obtain information on the system.

Some existing protocol composition frameworks offer these mechanisms. These mechanisms can be based on reflection techniques, provided by the meta-level architectures offered by the language in which they are implemented. Although well developed reflective mechanisms are used in different contexts [22,23], some even involving protocol compositions [24], their use is rudimentary in protocol composition frameworks, due to complex issues, s.a. protocol composition consistency and dependencies, or event flow. *Ensemble*, *Cactus*, and *Appia* frameworks allow to identify the protocols based on their names. *Samoa* framework supports the separation between the notion of protocol specification and protocol implementation but this is not enough when adaptation is not limited to the exchange of protocol implementations of the same protocol specification (the single adaptation action that is currently supported in *Samoa*).

4.2 Discussion

When discussing how the requirements are addressed by existing protocol composition frameworks, we have also identified that each requirement can be satisfied at a different level of abstraction. Some requirements may require specific support from the protocol composition framework runtime (for instance, the ability to change the composition in runtime). Other requirements can be satisfied by a number of complementary services that can be implemented on top of an existing composition frameworks. Finally, other requirements are better satisfied by enforcing a particular service programming model. We have observed that, although most of these requirements have been previously addressed by different frameworks, none of the existing composition framework satisfies completely the full set of requirements. Moreover, some of these requirements identified in the context of protocol composition frameworks also apply to component-based frameworks. However, these requirements have to be address in a different manner.

5 An Adaptation-Friendly Composition Framework

As a result of the previous analysis, we have implemented a service composition framework, named *RAppia*, that fulfills the set of requirements we have identified. This service composition framework has been built as an extension to one of the protocol composition framework surveyed: the *Appia* [5]. In the next paragraphs we describe the design and implementation of *RAppia*.

5.1 *RAppia* Basics

RAppia is a service composition framework implemented in the Java programming language. It inherits the composition model from the *Appia* protocol composition framework, that is common to many other similar frameworks (such as *x-kernel*, *Horus*, and *Ensemble*). In *RAppia* services can be composed in a layered manner, creating stacks of services. Typically, services at the bottom of a service composition offer more basic functionality (such as reliable multicast communication) and services at the top of the service composition support higher level abstractions (such as distributed shared object, publish-subscribe, etc).

An instance of a service composition is named a *service channel*. Each layer of a service channel is an instance of the corresponding service in the service composition. Thus, a service channel consists of a stack of service instances. Each instance maintains the state required to provide the desired service. Note that an application may create multiple service channels with the same composition (for instance, to maintain multiple shared objects).

Service instances interact through the exchange of events. Events in *RAppia* are object-oriented data structures. The *Event* class has two fundamental attributes: *channel*, and *direction*. The first is a reference to the service channel where the event will flow, and the second indicates in which direction the event is flowing along the service stack. Note that a session just forwards an event up or down in a channel, without having explicit knowledge of the concrete service that is executed above and below in the stack. This allows the stack to be reconfigured without changing the code of each service implementation.

When building distributed applications, many services are distributed. Furthermore, many services require the exchange of messages among different nodes. The information that needs to be sent over the wire is included in a special field of the events used for inter-service communication called a *Message*.

In *RAppia*, two or more service channels that share a given service may opt to share the same instance of that service. A shared service implementation may correlate events exchanged in different service channels with the help of locally maintained state.

Grounded on these basic mechanisms, the adaptation support is built considering three different aspects: the service programming model, adaptation-friendly services, and kernel mechanisms. These aspects are described next.

5.2 Service Programming Model

The adaptation requirements have been taken into consideration in the programming model used to implement services for *RAppia*. This has been reflected into three separate aspects: the set of events that need to be taken into consideration by each service implementation (which address *requirements 1 and 5*), how service properties are exposed (which addresses *requirement 6*), and how service implementation may exchange control information in a distributed setting (which is related to *requirement 5*).

Event Processing. In *RAppia* a service is implemented as a set of event handlers. In runtime, when events are delivered to a service, the appropriate handler is called. Typically, a handler does some processing and forwards the event to the next service in the composition. The framework does not restrict the type hierarchy of events that can be triggered and exchanged in the system. Still

RAppia defines a number of "system" events that should be handled by any service implementation. These include events to provide easy access to context information produced by the protocols (see *requirement 1*), events to handle state transfer and to place the service in a quiescent state (see *requirement 5*). More precisely, the following events are defined by *RAppia*:

– *ContextQuery*, *ContextAnswer*, and *ContextNotification* events. The first event is used to query a service for specific context information (such as the available bandwidth of a node at the present time), the second to reply to the query event(the reply with the bandwidth reading), and the later to allow a service to provide an asynchronous notification of context information (for instance, a drop in the bandwidth to zero). It is interesting to notice that although many composition frameworks define a number of mandatory events (for instance, an *Init* event used to initialize a service), to the best of our knowledge, no previous framework has been concerned with this sort of functionality, even if this is extremely relevant as these are basic services of any manageable object (from a systems' management perspective).
– *SetParameter* event. This event is used to update configuration parameters in runtime such as, for instance, timeout values.
– *MakeQuiescent* and *Resume* events. The first event is used to request a service to reach a quiescent state (as we have noted, often reconfiguration can only be performed if the service is in a quiescent state). This event is propagated in the channel in the *Down* direction. When the event reaches the bottom of the channel, its direction is reversed and when it reaches the top of the channel, the entire channel is in a quiescent state (as depicted in part of Figure 1). The second event, *Resume*, is used to resume the service after reconfiguration.
– *GetState* and *SetState* events. These events allow to transfer the service state from one instance to another, whenever the reconfiguration requires instances to be swapped (for instance, to install a software update). As illustrated in Figure 1, *GetState* event is propagated in the channel in the *Down* direction. When the event is received, each session adds a state object to the event, which includes all the state information to be transfered. The *SetState* event is propagated in the channel in the *Up* direction, after reconfiguration. Each session reads the corresponding state object and initializes its state variables accordingly.

Type Hierarchies. The definition of adaptation targets meta-information, namely for individual services and service channels, can be achieved through type hierarchies. The meta-information from services is defined based on the properties of the services such as: group communication, ordering, reliable, etc. Each service is tagged with the properties that it offers, from a well known set. The association of meta-information with service channels cannot be based in the same principle since channels with the same composition can be used for different purposes. Therefore, the meta-information is based on the type of task they perform, for example: control, audio, text, video, etc.

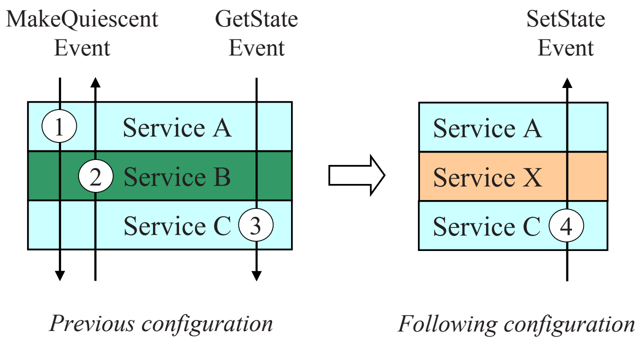


Fig. 1. Replacing service B by X: reaching quiescence and state transfer

The association of meta-information with services and service channels allows to define type hierarchies, based on the tag hierarchy. Therefore it will exist a hierarchy of service types and another of service channel types. These hierarchies are domain dependent, in the sense that applications with different domains may require different hierarchies. Further details on service type specification and hierarchies can be found in [25].

Message Headers. Most composition frameworks support a *message* abstraction that can be used by service implementations to exchange data with remote peers. In a service channel, each service may add/remove its own data to/from the message. The information added/removed by each service layer is typically called the *service header*.

There are two main approaches to manage service headers that have been implemented in existing protocol composition frameworks. One approach models the message as a stack of headers, exporting a push/pull interface to add/remove headers. This is the approach most widely adopted. Unfortunately, this solution is not very adaptation-friendly as it requires a strong coordination during reconfiguration (for instance, a header cannot be pushed unless the corresponding service is active in the remote node to perform the matching pull). Another approach, adopted in the *Cactus* [2] framework, consists in modelling the message as a *pool of headers*. This approach is more flexible, given that the header can be add/removed in different orders. *RAppia* adopted this approach.

Each header in the pool is identified by a textual label. The methods available to handle headers are "*addHeader(label,header)*", "*getHeader(label)*", "*removeHeader(label)*", and "*hasHeader(label)*". The method "*addHeader(label,header)*" adds a header associated with the given label; "*getHeader(label)*" reads the contents of the header associated with the given label; "*removeHeader(label)*" removes from the pool the header associated with the given label, and "*hasHeader(label)*" checks if the message contains the header with the given label. The management of the label namespace is orthogonal to the *RAppia* operation. However, *RAppia* requires each protocol to declare the labels of the headers it produces and requires, which mimics the *Appia* conventions to received and produced events. Therefore, the runtime can detect clashes in the header label namespace.

5.3 Adaptation-Friendly Services

RAppia includes two adaptation-friendly services: a generic and configurable context sensor (that addresses *requirement 2*) and a reconfiguration monitor (that addresses *requirements 4* and *5*). These services are described in the next paragraphs. Note that these services could also be adapted to be integrated in other composition frameworks, for instance, to *Cactus*.

Context Sensor. The context sensor is a service that is able to locally handle the capture of context information from running service compositions (as described in *requirement 2*). The context sensor is depicted in Figure 2, and works as follows.

The context sensor belongs to multiple service channels: a remote invocation channel, a context notification dissemination channel, and one or multiple sensed service channels, whose purpose is described below.

– The remote invocation channel is used to allow remote nodes to query context information on the sensed service channels. The context sensor receives context queries from this channel and forwards it to all sensed service compositions. Subsequently, it collects the correspondent context answers and sends back a reply on the sensor invocation channel.
– The context notification dissemination channel is used to disseminate to one or more remote nodes context notifications generated by any of the sensed compositions. The generic sensor simply intercepts any notification generated by one of the sensed compositions and forwards it to the notification dissemination channel. The sensor is oblivious to the composition of the notification dissemination channel. By selecting an appropriate dissemination channel, notification can be sent point-to-point to a centralized context monitor, in multicast to multiple nodes, or injected in a publish-subscribe infrastructure.
– The sensed service compositions channels are one or more channels whose context is locally monitored by the generic sensor.

Furthermore, the sensor can be also requested to perform periodic readings of on-demand readable context information and autonomously generate notification with a configurable period. Therefore, the sensor is prepared to, upon request, generate context notifications for variables that otherwise, would have to be read using explicit polling.

Finally, by carefully composing the notification dissemination channel, the programmer may easily introduce local processing at the sensed node to reduce network traffic. For instance, by adding a filter service to notification dissemination channel, one can prevent notifications, whose value is below a given threshold, to be disseminated to the network. In a similar manner, it is possible to include more sophisticated services in the notification channel, for instance, to compute the average of multiple notifications.

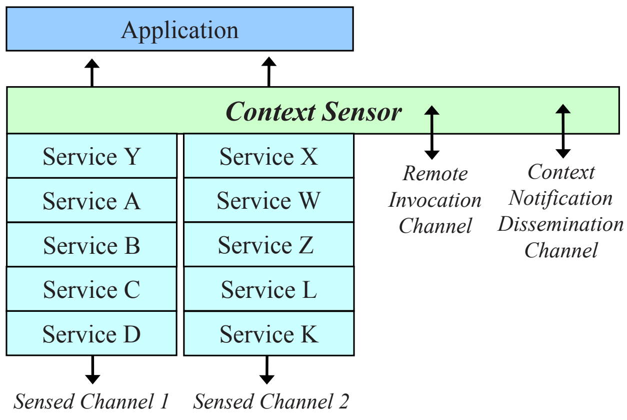


Fig. 2. Context sensor

Reconfiguration Monitor. The reconfiguration monitor is a service that interacts directly with the kernel of the composition service framework and exports a control channel through which it receives multiple *reconfiguration commands*. The reconfiguration monitor is depicted in Figure 3. Each reconfiguration command instructs the monitor to take one or more particular steps of a given reconfiguration sequence. The commands exported by the reconfiguration monitor are as follows.

– *MakeQuiescent*: this command instructs the monitor to put one or more services in a quiescent state, using the *MakeQuiescent* event.
– *Resume*: this command instructs the monitor to resume the activity of a service that was previously put in a quiescent state.
– *Store/LoadState*: these commands determine the capture of state information, and the loading in the end of the reconfiguration. For this purpose the monitor uses the *GetState* and *SetState* events.
– *Reconfigure*: this command instructs the monitor to reconfigure the composition of a given service channel. The reconfiguration involves one or more of the following actions: remove a service from the service channel, to add a service to a service channel, or to replace an instance of a service by an instance of an alternative service.

For more details on the reconfiguration monitor and the commands please refer to [13].

5.4 Kernel Mechanisms

To address *requirement 4*, the kernel of the *RAppia* composition framework includes two adaptation-friendly mechanisms that, to our knowledge, are not supported by any other composition framework: automatic buffering of events addressed to services in a quiescent state and automatic update of event routes, as described below.

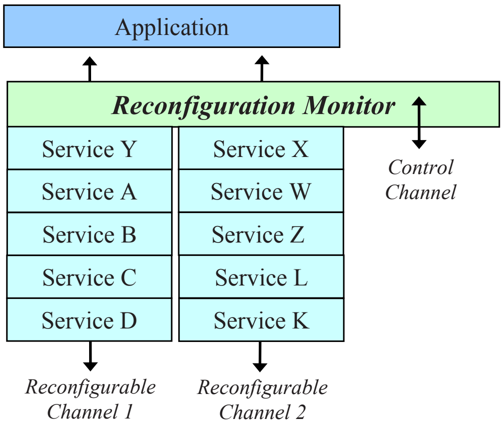


Fig. 3. Reconfiguration monitor

Event Buffering. As we have discussed previously, in order to reconfigure a service one may be required to put that service in a quiescent state. Typically, when in a quiescent state, the service is unable to process new events. Therefore, the *RAppia* kernel is able to recognize when a service is in a quiescent state and buffer all events addressed to that service. As soon as the service is resumed, the *RAppia* kernel restarts the delivery of events to the service. This functionality allows a service to be reconfigured without forcing the entire service channel to be put in a quiescent state.

Dynamic Update of Event Routes. The *RAppia* kernel is able to use information about which events are handled by each service to optimize the flow of events in a service composition. In particular, for each type of event, an event route is created. This ensures that an event is only delivered to the services that are interested in handling that event.

In an adaptive setting, the composition of a service channel may change in runtime. Furthermore, *RAppia* does not require the entire composition to be set in a quiescent state in order to perform the reconfiguration. Therefore, the *RAppia* kernel is built such that event routes are automatically recomputed when a reconfiguration occurs.

5.5 Discussion

We have implemented a prototype of the *RAppia* framework with the described features. This prototype is currently being used to build middleware systems for mobile networks, whose dynamic settings demand adaptation support. In this middleware, the *RAppia* adaptation-friendly services play an important role. Sensors can be configured to capture different context information, and the reconfiguration monitor allows to develop several different strategies to apply the reconfiguration actions, that are tailored to the service being reconfigured. Moreover, these mechanisms allowed us to build both a context monitor (to reason about context information), and an adaptation manager to control the adaptation process. A detailed description of these additional middleware components is outside the scope of this paper (the interested reader is referred to [13]).

6 Conclusions

Service composition frameworks are a significant component of any adaptive middleware infrastructure. Given the large experience in the design and implementation of composition frameworks oriented for communication protocols, it is interesting to use them as the basis for a adaptation-friendly service composition framework. This paper has identified a set of requirements imposed by adaptive middleware on composition frameworks. Subsequently, we have analyzed how these requirements have already been addressed in the context of protocol composition frameworks. Based on this analysis we propose an adaptive friendly service composition framework that has been obtained by extending an existing protocol composition framework with an augmented programming model, new adaptive services and a set of adaptation-friendly kernel mechanisms.

Acknowledgments

The authors are grateful to the anonymous referees for their comments on a previous version of this paper. This work was partially funded by FCT project MICAS – Middleware for Context-aware and Adaptive Systems – (POSI/EIA/60692/2004) through POSI and FEDER.

References

1. Hutchinson, N.C., Peterson, L.L.: The x-kernel: An architecture for implementing network protocols. IEEE Trans. Softw. Eng. 17(1), 64–76 (1991)
2. Hiltunen, M.A., Schlichting, R.D., Ugarte, C.A., Wong, G.T.: Survivability through customization and adaptability: The cactus approach. discex 01, 294 (2000)
3. van Renesse, R., Birman, K.P., Maffeis, S.: Horus: a flexible group communication system. Communications ACM 39(4), 76–83 (1996)
4. Cadot, S., Kuijlman, F., Langendoen, K., van Reeuwijk, K., Sips, H.: Ensemble: A communication layer for embedded multi-processor systems. In: LCTES 2001. Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems, pp. 56–63. ACM Press, New York (2001)
5. Miranda, H., Pinto, A., Rodrigues, L.: Appia, a flexible protocol kernel supporting multiple coordinated channels. In: ICDCS-21. Proceedings of The 21st International Conference on Distributed Computing Systems, pp. 707–710. IEEE Computer Society Press, Los Alamitos (2001)
6. Brasileiro, F., Greve, F., Tronel, F., Hurfin, M., Narzul, J.P.L.: Eva: An event-based framework for developing specialized communication protocols. In: NCA 2001. Proceedings of the IEEE International Symposium on Network Computing and Applications, pp. 108–120. IEEE Computer Society Press, Los Alamitos (2001)

7. Wojciechowski, P., Rütti, O., Schiper, A.: SAMOA: A Framework for a Synchronisation-Augmented Microprotocol Approach. In: IPDPS 2004. 18th International Parallel and Distributed Processing Symposium, vol. 01, pp. 64–74. IEEE Computer Society, Los Alamitos (2004)
8. Rosa, L., Rodrigues, L., Lopes, A.: Building adaptive services for distributed systems. Technical report, Dept. Informatics, University of Lisbon (2007)
9. Rütti, O., Wojciechowski, P.T., Schiper, A.: Service interface: a new abstraction for implementing and composing protocols. In: SAC 2006. Proceedings of the 2006 ACM symposium on Applied computing, pp. 691–696. ACM Press, New York (2006)
10. Garbinato, B., Guerraoui, R.: Flexible protocol composition in bast. In: ICDCS 1998. Proceedings of the The 18th International Conference on Distributed Computing Systems, pp. 22–30. IEEE Computer Society Press, Los Alamitos (1998)
11. Chen, W.K., Hiltunen, M.A., Schlichting, R.D.: Constructing adaptive software in distributed systems. In: ICDCS 2001. Proceedings of the The 21st International Conference on Distributed Computing Systems, pp. 635–643. IEEE Computer Society Press, Los Alamitos (2001)
12. van Renesse, R., Birman, K., Hayden, M., Vaysburd, A., Karr, D.: Building adaptive systems using ensemble. Softw. Pract. Exper. 28(9), 963–979 (1998)
13. Rosa, L., Rodrigues, L., Lopes, A.: A framework to support multiple reconfiguration strategies. Technical report, Dept. Informatics, University of Lisbon (2007)
14. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Technical report, Hanover, NH, USA (2000)
15. Acharya, A., Ranganathan, M., Saltz, J.H.: Sumatra: A language for resource-aware mobile programs. In: Tschudin, C.F., Vitek, J. (eds.) MOS 1996. LNCS, vol. 1222, pp. 111–130. Springer, Heidelberg (1997)
16. Kwon, Y., Fang, Y., Latchman, H.: Performance analysis for a new medium access control protocol in wireless lans. Wirel. Netw. 10(5), 519–529 (2004)
17. Kwon, Y., Fang, Y., Latchman, H.: Improving transport layer performance by using a novel medium access control protocol with fast collision resolution in wireless lans. In: MSWiM 2002. Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems, pp. 112–119. ACM Press, New York (2002)
18. Ketfi, A., Belkhatir, N., Cunin, P.Y.: Automatic adaptation of component-based software: Issues and experiences. In: PDPTA 2002. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 1365–1371. CSREA Press (2002)
19. Liu, H.: A component-based programming model for autonomic applications. In: ICAC 2004. Proceedings of the First International Conference on Autonomic Computing, pp. 10–17. IEEE Computer Society Press, Los Alamitos (2004)
20. Crawley, S., Davis, S., Indulska, J., McBride, S., Raymond, K.: Meta information management. In: FMOODS 1997. Proceeding of the IFIP TC6 WG6.1 International Workshop on Formal Methods for Open Object-based Distributed Systems, pp. 193–202. Chapman & Hall, Ltd, Sydney (1997)
21. da Silva e Silva, F.J., Kon, F., Yoder, J., Johnson, R.: A pattern language for adaptive distributed systems. In: SugarLoafPLoP 2005. Proceedings of the 5th Latin American Conference on Pattern Languages of Programming, Campos do Jordão, Brazil, pp. 19–48 (2005)
22. Chiba, S., Masuda, T.: Designing an extensible distributed language with a meta-level architecture. In: Nierstrasz, O. (ed.) ECOOP 1993. LNCS, vol. 707, pp. 482–501. Springer, Heidelberg (1993)

23. Fabre, J., Nicomette, V., Perennou, T., Stroud, R., Wu, Z.: Implementing fault-tolerant applications using reflective object-oriented programming. Technical report (1995)
24. Agha, G., Frølund, S., Panwar, R., Sturman, D.: A linguistic framework for dynamic composition of dependability protocols. In: Dependable Computing and Fault-Tolerant Systems VIII, IFIP Transactions, pp. 345–363. Springer, Heidelberg (1993)
25. Rosa, L., Lopes, A., Rodrigues, L.: Policy-driven adaptation of protocol stacks. In: ICAS 2006. Proceedings of the International Conference on Autonomic and Autonomous Systems, pp. 5–12. IEEE Computer Society Press, Los Alamitos (2006)

Invasive Patterns for Distributed Programs*

Luis Daniel Benavides Navarro, Mario Südholt,
Rémi Douence, and Jean-Marc Menaud

OBASCO project; EMN-INRIA, LINA
Dépt. Informatique, École des Mines de Nantes
4 rue Alfred Kastler, 44307 Nantes cédex 3, France
{lbenavid, sudholt, douence, jmenaud}@emn.fr

Abstract. Software patterns have evolved into a commonly used means to design and implement software systems. Programming patterns, architecture and design patterns have been quite successful in the context of sequential as well as (massively) parallel applications but much less so in the context of distributed applications over irregular communication topologies and heterogeneous synchronization requirements.

In this paper, we propose a solution for one of the main issues in this context: the need to complement distributed patterns with access to execution state on which it depends but that is frequently not directly available at the sites where the patterns are to be applied. To this end we introduce *invasive patterns* that couple well-known computation and communication patterns like pipelining and farming out computations with facilities to access non-local state. We present the following contributions: (i) a motivation for such invasive patterns in the context of a real-world application: the JBoss Cache framework for transactional replicated caching, (ii) a proposal of language support for such invasive patterns, (iii) a prototypical implementation of this pattern language using AWED, an aspect language for distributed programming, and (iv) an evaluation of our proposal for refactoring of JBoss Cache.

1 Introduction

Software patterns have proven a versatile tool for program development, be it for the development of application designs [15], architecture descriptions [24] or program implementations [14]. Design patterns have been very successful in the domain of sequential, in particular object-oriented applications. Similarly, pattern-based development methods have been extensively applied in the parallel domain for the derivation and implementation of massively parallel algorithms [24,21,9]. However, pattern-based approaches have been much less successful in the domain of distributed programming, in particular, if they are defined over irregular communication topologies and subject to heterogeneous synchronization constraints. Consequently, patterns for distributed programming

* Work partially supported by AOSD-Europe, the European Network of Excellence in AOSD (www.aosd-europe.net)

(see, for instance, patterns for distributed enterprise information systems and grid applications [20,10,14]) are often expressed as mere programming recipes that are not backed up by concrete architecture or implementation entities that can be used and reused as building blocks for applications.

In this paper we investigate a major reason for the difficulty in applying programming patterns, that embody common computation and communication patterns, to distributed applications: frequently, applications of such patterns in realistic contexts depend on information on the execution state that is not directly available when the pattern is to be applied. This is, for instance, the case in two frequent cases: (i) in legacy contexts where patterns could be used to improve the application structure but in which instructions for communication instructions and manipulation of related execution state are frequently scattered over numerous places and (ii) distributed applications that have been designed using less flexible abstractions than provided by communication and computation patterns.

In this paper we introduce *invasive patterns* for distributed programming. Such patterns essentially provide well-known regular computation and communication patterns but provide a built-in abstraction for access to non-local execution state whose access is required to enable pattern applications. We provide evidence that techniques from Aspect-Oriented Programming (AOP) [1] can be harnessed to augment patterns by structure access to such non-local state.

Concretely we present the following contributions. First, we present a detailed motivation for invasive patterns and corresponding aspect-oriented support based on a detailed analysis of the use of patterns in a real-world distributed application: the JBoss Cache strategy for replication in the context of transactions. Second, we introduce a pattern language that allows to concisely define invasive variants of well-known patterns for distributed applications. Third, we briefly sketch a prototypical implementation of invasive patterns using the AWED system for explicit distributed aspect-oriented programming. Fourth, we give an evaluation of our approach by discussion how invasive patterns can be used to improve the structure of JBoss Cache.

The paper is structured as follows. In Sec. 2, we introduce the notion of invasive patterns and motivate it in a real-world application. Our pattern language is introduced in Section 3. In Sec. 4, we describe our prototypical implementation of this pattern language using AWED. Sec. 5 presents the evaluation of our approach. Related work is discussed in Sec. 6. Finally, Sec. 7 gives a conclusion and discusses future work.

2 Motivation

In this section we first present modularization problems of pattern-like computations in JBoss Cache [17], a large-scale real-world framework for transactional replicated caching. We then introduce the notion of invasive patterns that we propose as a means to resolve such modularization problems.

2.1 Pattern-Like Structures in JBoss Cache

We have analyzed the occurrences of pattern-like computation structures and the dependencies of such pattern-like structures on the underlying execution state in JBoss Cache, an open source implementation of a replicated transactional cache over an J2EE-based communication infrastructure. In the following we briefly describe the JBoss Cache framework and the results of our analysis of software patterns that are used implicitly in this infrastructure.

JBoss Cache is a large object-oriented framework implemented in Java that consists of more than 50 thousand lines of code. Basically the JBoss Cache implementation consists of two main parts: (i) a main class `TreeCache` that represents the main data structure, a tree with a hash table on each leaf, that is replicated on each node (host) in the cache cluster and (ii) a set of filters that is used to implement the major part of the behavior of non-functional requirements, mainly transactions and data replication. An interception mechanism is used to transfer control between the classes implementing the data structure and the filters. Concretely, each call to the `TreeCache` API is first transformed into a method call object using a reflection mechanism. Once this object is created, it is passed to a chain of filters where each filter adds some behavior, *e.g.*, optimistic locking is added by the transaction filter. Eventually, the filtered method call is performed.

The current production version (1.4) of JBoss Cache conceptually uses an architecture that can be expressed nicely in terms of patterns using, *e.g.*, a pipeline pattern for transaction control and a farm pattern for replication actions (Here and in the rest of the paper we assume the cache to be configured for transactions with pessimistic locking and a two phase commit protocol). Figure 1 presents a high-level pattern-based view of the corresponding system structure of JBoss Cache. In the figure, a transaction is triggered by a specific method call represented by the first node in the pattern. Then successive calls to `get`, `remove` or `put` methods on the cache are executed and the information is stored for further replication. When a particular value is not present in the cache, the cache looks for the value in a group of selected neighboring nodes, its so-called buddies, illustrated by the three edges starting in the second node of the figure. Once the end of a transaction is reached, the originating cache engages a two phase commit protocol. In such a protocol the originating cache sends a prepare message with the transaction control information (edges numbered 1 in the right part of the figure), followed by answers from all buddies confirming agreement or non agreement (edges numbered 2). Finally, the originating cache sends a final commit or a rollback message depending on the answers it received (edges numbered 3). Note that in this interaction we can identify, at least, two well separated groups of hosts, one for the search of values at buddy nodes and the other for the replication behavior from a node to other nodes.

In previous work [6] we have analyzed the complexity of the code structure of the (non-pattern based) implementation of the JBoss Cache framework: we have shown that replication and transaction instructions are, in particular, widely

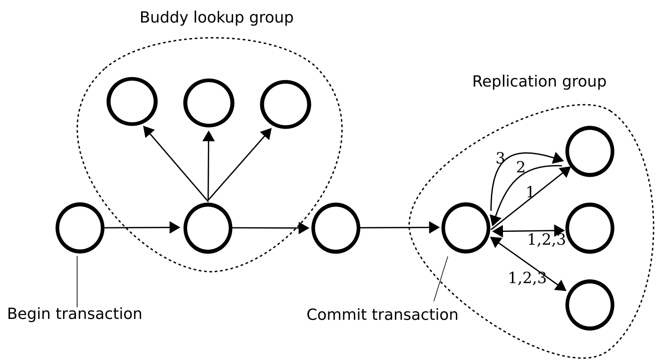


Fig. 1. Architecture of transaction handling with replication in JBoss Cache

scattered over the code base and tangled with one another in numerous places.[1] Even though JBoss Cache conceptually is characterized by a pattern-based structure as shown in Fig. 1, the current implementation does not allow conventional patterns for distributed systems to be applied due to the scattering and tangling of code these functionalities are subject to. The `TreeCache` class consists of 3802 lines of code (LOC), of which more than 280 LOC are relevant for transactions. The interceptor package exhibits similar quantitative characteristics: the package consists of 5099 LOC and more than 137 LOC are related to transactional behavior and are not included in the dedicated transaction interceptor. A detailed qualitative analysis of such code leads to the identification of three basic problems:

1. Transactional and replication behavior depends on state that is stored in different classes. Such state is modified in scattered pieces code that, *e.g.*, reify the current transactional state as mentioned above so that it can later be tested in another class in order to decide which replication action to perform.
2. The relationships governing the interplay between the main concerns, transactions and replication, are not made explicit anywhere in the code. Instead, scattered pieces of code implicitly coordinate these concerns, thus generating tangled code and breaking the modularization aimed at by the JBoss Cache filter mechanisms.
3. JBoss cache includes several distribution-related concerns (*e.g.*, replication, cache loaders and buddy lookup) that require communication between different groups of hosts. Group overlapping and interactions between different groups generate additional tangling.

2.2 Source Code Representation of Pattern-Like Structures

These problems are clearly apparent in the source code of JBoss Cache. In abstract terms, a cache behaves as follows. A chain of interceptors for the support

[1] This analysis has been conducted on JBoss Cache version 1.2 but remains valid for the current production version 1.4 as shown in [23].

of transactional and replication behavior is created when the cache is initialized. When a transaction-related action occurs, a method-call object of a corresponding type is created using the JAVA reflection framework, which is then passed through the chain of interceptors. A ”get” request, for example, may be processed by filter to check its buddies if a specific data is in their cache, by the transactions interceptor to control transactional behavior, and by the locking interceptor to lock the tree cache data structure accordingly. The so-called replication interceptor, finally, performs (most of) the two-phase commit protocol among caches by first sending a *prepare* message, followed by a *rollback* or *commit* message depending on the result of the prepare phase. This code architecture is problematic because the manipulation of state that is relevant for replication operations as well as the protocols governing transactional and replication behavior is determined by scattered pieces of code whose joint effects during execution, *i.e.*, the correct implementation of transactional behavior and replication, are very difficult to apprehend from the code structure.

Figure 2 shows a piece of code of the main filter method `invoke` of the `DataGravitationInterceptor` class that is responsible for the so-called data gravitation concern, *i.e.*, buddy lookup. This method clearly exhibits the problems stated above, providing evidence of tangling of three concerns: replication, transactions and buddy lookup. The code uses a common idiom to address transactions control inside a `switch` instruction (lines 7 to 21). The right branch in the switch statement is taken depending on static information on the execution state, *e.g.*, a configuration-time choice between optimistic and pessimistic locking, and dynamic information about the execution state, *e.g.*, the dynamic type of the current processed method call. There, in order to calculate the method id (see line 5) the application relies on an ad-hoc mapping that is defined in the class `MethodDeclarations`. Similarly, the choice between optimistic and pessimistic locking is made at configuration time inside the `TreeCache` class as well as part of the class `InterceptorChainFactory` (this choice in turn affects at runtime the configuration of the dynamically created chain of filters).

Note that the corresponding piece of code is found inside the filter class `DataGravitation` and uses data that is calculated in many different places, thus expliciting the problem 1 above. The kind of idiom involving `switch` statements (that clearly represent a mismatch between the conceptual pattern-based architecture and its concrete implementation) is scattered over multiple places in the implementation. We have found 93 places where such a switch action is used and more than 29 places where it occurs in the context of replication operations implying a *one to many* communication between caches (thus providing testimony for the problems 1 and 2 introduced above).

Furthermore, the `DataGravitation` class plays an unexpected role in the two phase commit protocol. A method of type `commitMethod` is processed in order to send a commit message on those caches that are not part of the current buddy group, see line 37 in the `docommit` method (*i.e.*, being subject to problem 3 above). Remember that the `DataGravitation` class was supposed not to control the transactional behavior or the replication of transactions which, should normally, be performed by the transactions and replication filters.

```
//----- Piece of code in the invoke method of
//----- DataGravitationClass
try
{
 switch (m.getMethodId())
  {
    case MethodDeclarations.prepareMethod_id:
    case MethodDeclarations.optimisticPrepareMethod_id:
         Object o = super.invoke(m);
         doPrepare(getInvocationContext().getGlobalTransaction());
         return o;
    case MethodDeclarations.rollbackMethod_id:
         transactionMods.remove(
                 getInvocationContext().getGlobalTransaction());
         return super.invoke(m);
    case MethodDeclarations.commitMethod_id:
         doCommit(getInvocationContext().getGlobalTransaction());
         transactionMods.remove(
                   getInvocationContext().getGlobalTransaction());
         return super.invoke(m);
  }
}
catch (Throwable throwable)
{
 transactionMods.remove(
                 getInvocationContext().getGlobalTransaction());
 throw throwable;
}

//---- The docommit method in DataGravitation class
private void doCommit(GlobalTransaction gtx) throws Throwable
   {
      if (transactionMods.containsKey(gtx))
      {
         if (log.isTraceEnabled())
               log.trace("Broadcasting commit for gtx " + gtx);
         replicateCall(getMembersOutsideBuddyGroup(),
            MethodCallFactory.create(
                  MethodDeclarations.commitMethod,
                  new Object[]{gtx}),
                  syncCommunications);
      }
      else
      {
         if (log.isTraceEnabled())
             log.trace(
              "Nothing to broadcast in commit phase for gtx " + gtx);
      }
   }
```

Fig. 2. Tangled code of a two phase commit (2PC) protocol inside the invoke method of the `DataGravitationInterceptor` class

2.3 Invasive Patterns in a Nutshell

Dependencies as those motivated above for JBoss Cache between transaction-related actions and replication operations cannot simply be modularized using standard patterns for workflow-related computations, such as pipelining, farming out or gathering computations as illustrated in Fig. 3, where circles denote calculations that possibly take place on different hosts and edges denote communication. In fact, taking scattering and tangling of transactions and replication

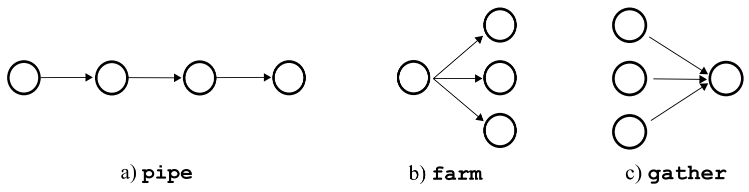


Fig. 3. Basic patterns

into account requires does not fit the common interpretation of such patterns in which each circle denotes a well-defined entity, in our motivating example some nicely modularized piece of code within JBoss Cache.

In such cases effective support for a pattern-based programming style should allow the definition of patterns to include accesses to the data it depends on but that is defined at other places in the underlying distributed program and allow such patterns to be applied possibly at numerous places in a program. Because crosscutting of non-local execution state that enables such pattern applications is at the heart of such effective support, Aspect-Oriented Programming [18,1] seems a promising approach for the modularization of patterns and the corresponding data accesses.

We pursue this idea in this paper on the programming level by extending patterns with a notion of aspects to modularize such crosscutting accesses. The resulting notion of invasive patterns is illustrated in Fig. 4 for the case of a `gather` pattern. On the three nodes on the left hand side, different pointcuts (represented by dashed lines) are used to access information that is then prepared by "source" advice (represented by the filled rectangles) to be sent to the right hand side node. Once all relevant data has been passed to the right hand side

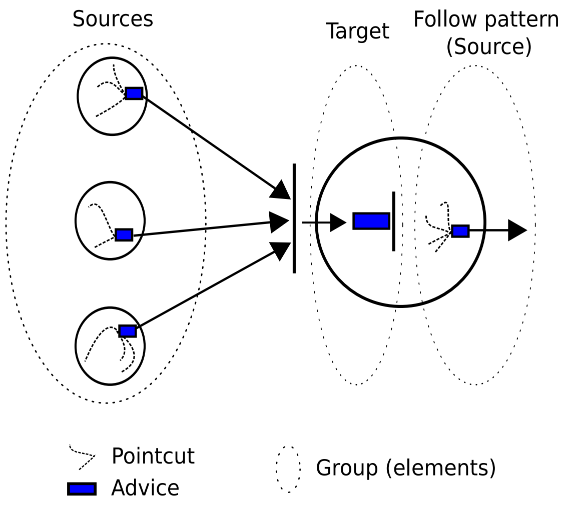


Fig. 4. Invasive patterns

node, a "target" advice is used to integrate the transmitted data with an existing or new computation on the target node. In order to support the declarative definition of such crosscutting accesses, we leverage results on so-called stateful pointcut languages [13] that enable matching of sequences of execution events to be defined using expressive languages, in particular finite-state automata.

Besides a definition of basic invasive patterns a suitable notion of pattern composition is needed. Reconsider the (abstract) architecture of transaction handling with replication in JBoss Cache, see Fig. 1: this architecture can naturally be expressed in terms of compositions of the three basic patterns introduced above, where the steps denoted 1–3 in the figure correspond, for instance, to two applications of the `farm` pattern and one application of the `gather` pattern. Our approach supports the compositional construction of such architectures from the basic patterns on the programming and the implementation level. As discussed in Sec. 2.1, this architecture is essentially hidden in the actual JBoss implementation. Our approach can therefore be seen as a means to make explicit such architectures, and thus help program understanding and maintainability.

3 Pattern Language

A crucial issue concerning invasive patterns as motivated before is how the different activities (pointcut matching, local and remote advice) are synchronized with one another. In this section, we first discuss corresponding design choices and then present our language for the definition of invasive architectural patterns.

3.1 Design Choices

The definition of distributed algorithms using patterns over a state-based programming paradigm essentially depends on the correct synchronization on the different parts of invasive patterns and between different invasive patterns. Pattern-based computations can be synchronized roughly at three different levels:

1. *Synchronization within an invasive pattern.* Most basically, target advice is executed only after a rendez-vous synchronization of all source computations. In the case of the `gather`-pattern shown in Fig. 4, the target computation is started only after the three target hosts have "agreed" to trigger it.
   Second, target advice may be executed in a synchronous or asynchronous fashion. Synchronous execution of parts of the `pipe` pattern of Fig. 3a corresponds to a fully sequential (a.k.a. batch) computation, while its asynchronous execution corresponds to a pipelined computation. We support both behaviors.
2. Computations involving *consecutive executions of patterns* may be synchronized with one another. The `gather` pattern may, for instance, be synchronized with the execution of the following pattern that is represented in the right hand side node by the pointcut (the dotted lines), the source advice (the small rectangle) and the arrow leaving the node to the right. Execution of a follow pattern on a node n must obviously start after control of the previous pattern has entered n (otherwise the two pattern executions could not

be said to be consecutive) but may be reasonably started either when the target advice of the previous pattern is started or when it terminates. In this paper we only consider the synchronous case, *i.e.*, execution of follow patterns start when the target advice finishes. Our prototype implementation already supports both options, though.

3. Most generally, synchronization constraints may be imposed on *arbitrary segments of pattern compositions*. Such general constraints are interesting, *e.g.*, because computations may be executed on the same host and therefore give rise to problems, such as race conditions. Such synchronization strategies cannot, however, be defined simply in terms of individual patterns as considered here.

Summarizing, we provide in this paper explicit support for intra-pattern synchronization and synchronization between consecutive pattern executions. We do not, however, provide general synchronization strategies over pattern compositions because they are difficult to comprehend and may easily lead to performance bottlenecks or even deadlocks. We envision that specific properties over pattern compositions can be analyzed and enforced in terms of the more restricted means for synchronization we introduce here. This issue is, however, beyond the scope of the present paper.

3.2 Syntax and Informal Semantics

We are now ready to introduce the pattern language we have designed that realizes the above design choices. Figure 5 shows the syntax of our pattern language (we have omitted details for the sake of simplicity).

The pattern constructor `patternSeq` takes as argument a list $G_1\ A_1\ G_2\ A_2 \ldots G_n$ of alternating group and aspect definitions. Each triple $G_i\ A_i\ G_{i+1}$ in this list corresponds to a pattern application that uses the aspect A_i to trigger the pattern in a source group G_i and realize effects in the set of target hosts G_{i+1}. A group G is either defined as a set of host identifiers H or through a pattern constructor term itself. In the latter case, the group is defined as the source or target group of the constructor term depending on the argument position the term is used in. This constructor enables to define the basic patterns shown in Fig. 3: `pipe` as a `patternSeq` from a single host to another, `farm` as a `patternSeq` from a single host to several hosts and `gather` as a `patternSeq` from several hosts to a single one. Pattern compositions can be defined with more complex `patternSeq` terms. For instance, the left hand side of Figure 6 defines a

$$\begin{array}{lll}
P & ::= & \texttt{patternSeq}\ G_1\ A_1\ G_2\ A_2 \ldots G_n \\
G & ::= & H\ G \quad |\ P\ G \quad |\ \epsilon \\
A & ::= & \texttt{aspect}\ \{\ \texttt{around}((H, \mathrm{Id}^*)^*)\!: \mathit{PCD} \quad \mathit{SourceAdvice} \quad [\texttt{sync}]\ \mathit{TargetAdvice}\ \} \\
\mathit{PCD} & ::= & \texttt{call}(\mathit{MSig}) \quad |\ \texttt{target}(\mathit{Id}) \quad |\ \texttt{args}(\mathit{Id}+) \\
 & | & \mathit{PCD}\ \&\&\ \mathit{PCD} \quad |\ \mathit{PCD}\ ||\ \mathit{PCD} \quad |\ !\mathit{PCD} \\
 & | & \mathit{Seq}
\end{array}$$

Fig. 5. Pattern language

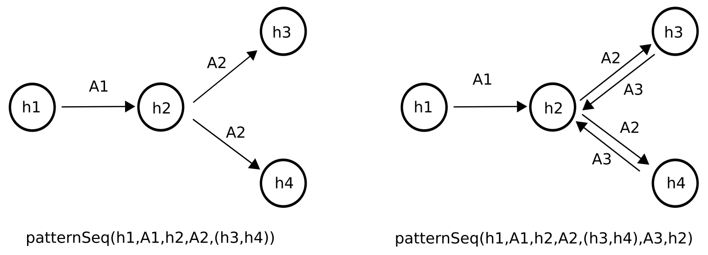

h1
A1
h2
A2
h3
A2
h4
patternSeq(h1,A1,h2,A2,(h3,h4))
h1
A1
h2
A2
A3
h3
A2
A3
h4
patternSeq(h1,A1,h2,A2,(h3,h4),A3,h2)

Fig. 6. Pattern Compositions

composition `pipe` then `farm`, and its right hand side defines a composition `pipe`, `farm` then `gather`. These examples make clear it is easy to define sophisticated compositions akin to the architecture of transaction handling in JBoss Cache (cf. Fig. 1).

Aspects A that define the behavior of invasive patterns specify a pointcut PCD that allows to modularize crosscutting code that triggers a pattern, and define a source advice and a target advice executed respectively on the source and target groups of a pattern. Advice can be parametrized by source hosts H and bound values (see `args` below). An advice is a standard block for code, but a source advice can call the matched base call with the `proceed` keyword. Otherwise, the base call triggers the aspect but the execution of the corresponding base method is skipped. When a `sync` annotation is used to qualify target advice, the base program execution on source hosts is not resumed before the end of the target advice. The default behavior is asynchronous execution.

We consider pointcut definitions that, for presentation purposes, are essentially restricted to matching of method call joinpoints, may extract target objects with `target` and arguments of calls with `args` and use logical compositions of pointcuts. Following the paradigm of stateful pointcuts [13,6] (and unlike AspectJ [4,19]) pointcuts may match sequences (non-terminal *Seq*) of calls in the base program execution. We omit the syntax of sequences for now, but they are basically defined in terms of a finite-state automaton by declaring its states and by labelling state transitions with pointcuts.

Let us consider a small example. The aspect in Figure 7 profiles session creation. When the method `login` is called the local advice performs it (through a call to `proceed()`) and the target advice increments the integer counter defined within the aspect. This aspect can be applied using `patternSeq` to two hosts so that sessions on the first host are counted on the second.

```
aspect Profiling {
  int sessions=0;
  around(): call(* *.login()) { proceed(); } { sessions++; }
}
```

Fig. 7. A Session Profiling Aspect

4 Implementation

In order to implement the pattern language presented in the previous section, support for three main mechanisms is necessary: (i) aspects providing a modular abstraction for invasive access on the source hosts and triggering activities on target hosts, (ii) flexible means for synchronization within individual patterns and between consecutive pattern executions, and (iii) the concise definition of the communication topologies of patterns.

Mainstream sequential AOP languages, in particular AspectJ [19], do not fit well these requirements because they do not include any specific support for distribution and concurrency and are therefore subject to well-known deficiencies if used for the modularization of distribution concerns (as exposed, *e.g.*, by Soares et al. [22]). Concretely, with regard to invasive patterns such aspect languages would require to split the definition of patterns into different, at the application-level unrelated aspects that have to be manually deployed on different hosts.

We have implemented invasive patterns using a recent approach to AOP for distributed applications, Aspects with Explicit Distribution (AWED) [6,5], which provides direct support for most of the necessary features and allows to accommodate the remaining ones based on its native abstractions. The AWED language has been designed as an aspect language for the modularization of crosscutting concerns in distributed systems. In general terms, AWED allows to define pointcuts that match sequences of execution events on different hosts in a distributed systems that trigger advice that is executed on potentially different hosts.

Figure 8 illustrates the two main features of the language: remote pointcuts and advice. Pointcuts essentially allow to match sequences of execution events that occur on different hosts. Hosts can be referred to using absolute addresses but can also be defined relative to the host on which an aspect is deployed (term `localhost`, in the figure the host colored in gray). Remote advice can be triggered on other hosts using the `on` specifier. Besides the host specifications available for pointcut definitions, advice execution can also be specified to take place on the host where the pointcut has been matched (term `jphost`). Pointcuts

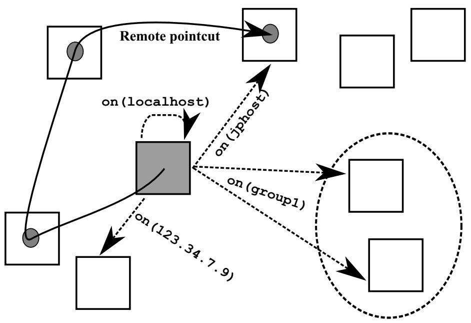


Fig. 8. Remote pointcuts and advice in AWED

and remote advice execution may depend on explicitly defined groups of hosts. In pointcuts, such groups may limit matching of execution events to sets of hosts; as to advice executions, groups allow to execute advice on several hosts. Furthermore, AWED allows to execute pieces of advice synchronously or asynchronously with the execution of the base application and with other aspects.

A farm pattern can be mapped to an AWED aspect having a pointcut expression as

`call`(* *.login()) && `host`(”sources”) && `on`(”targets”),

there, the `call` pointcut matches calls to *login* method. The pointcut `host`(”sources”) matches the join points (events) that appear in a host that belongs to the *sources* group. Finally the pointcut `on`(”targets”) triggers the execution of the advice in hosts that belong to the *targets* group. AWED also supports the `Seq` pointcut that allows to specify finite-state automata that permit to match sequences of join points in distributed applications. The sequence constructor is used to map direct uses of *Seq* pointcuts of our pattern language and to implement *rendez-vous* synchronization in gather-like patterns. We have developed a formally-defined transformation from our aspect language into executable AWED programs.[2] More information on the concrete translation of programs expressed using the pattern language into AWED programs can be found along with substantial examples in the following evaluation section.

5 Evaluation

In this section we evaluate our approach by presenting how invasive patterns can be used to restructure transaction handling and replication in JBoss Cache We first how to implement these concerns using the proposed pattern language, thus making explicit their pattern-based structure. We then briefly discuss the resulting implementation in AWED. Third, we qualitatively evaluate the resulting pattern-based implementation by discussion the difference in conciseness of the original and new implementation. Finally, we briefly discuss first benchmarking results we have performed by executing the refactored implementation of JBoss Cache using the current AWED implementation [5].

5.1 JBoss Cache Revisited

Invasive patterns allow to concisely express the essentials of the pattern-based architecture for transaction handling and replication in JBoss Cache as shown in Fig. 1. Concretely, we have implemented support for transactions with pessimistic locking and the two phase commit protocol using invasive patterns.

The corresponding solution is formulated in terms of a nested composition involving four pattern expressions, see Fig. 9. First, we apply a `pipe` pattern to be able to relate the start of transactions with the replication operations, *i.e.*, the

[2] Note to reviewers: this formal transformation, that cannot be described here because of lack of space, is available on request.

```
      gCaches = {H1, H2, H3}
      pipe([h],
           Atransac,
           farm(
             gather(
               farm([h], Aprepare, sync gCaches-[h]),
               Apresp,
               [h]),
             Acommit,
             gCaches-[h])
          );
```

Fig. 9. Pattern-based definition of the JBoss Cache two phase commit

start node and the final replication group, respectively, of Fig. 1. Once a commit is encountered, a `farm` pattern is used to farm-out the prepare phase of the two phase commit protocol. Then, a `gather` pattern is used to collect the answers from the involved buddy caches. Finally, after all answers have been received we use again a `farm` pattern to distribute the final decision of commit or rollback. The code in the figure defines this algorithm for three replicated caches. Note that replication can be triggered from any of the three caches. Once the triggering node (`h` in the algorithm) is selected the expression `gcaches-h` represents the group of caches without the triggering one.

Figure 10 shows the pattern-defining aspect `Aprepare` that farms out the prepare information of the two phase commit protocol. Occurences of calls to the `prepare` method are matched and executed (because of the call to `proceed` in the source advice). On the target hosts, the target advice executes the prepare phase followed by the invocation of an agreement or disagreement method, depending of the answer of the target caches. The aspect takes care of transactions that

```
aspect Aprepare {
  org.jboss.cache.TreeCache tc = CacheRegistry.getInstance().getCache();

  around(DataStorage d, String txId):
    call(* PrepareHelper.send(..)) && args(d,s) &&
    !cflow(call(TransactionManager.prepare(..)))

    // Source advice
    { proceed(); }

    // Target advice
    { TransactionManager tm = TransactionManager.getInstance();
      PrepareHelper ph = new PrepareHelper();
      try{
          tm.prepare(d, txId, tc);
        ph.respAgree(txId);
      } catch(Exception e) {
        ph.respNotAgree(txId);
      }
    }
}
```

Fig. 10. 2PC invasive aspect `Aprepare`

```
all aspect Aprepare_AWED {
  org.jboss.cache.TreeCache tc = CacheRegistry.getInstance().getCache();

  Group[] targetGs = {new Group("h1"), new Group("h2"), new Group("h3")};

  pointcut sourcePrepareCall(TransactionData d, String txId):
        seq(init:call(* Atransac.triggerNext()),
        pcd: call(* PrepareHelper.send(..)));

  pointcut targetPrepareCall(Transaction tx)(TransactionData d, String txId):
                call(* PrepareHelper.send(..));

 // source advice
 around(TransactionData d, String txId): sourcePrepareCall(d, txId) && host(localhost) {
   proceed();
 }

  // target advice
  after(TransactionData d, String txId): targetPrepareCall(tx) && on(targetGs) {
    TransactionManager tm = TransactionManager.getInstance();
    PrepareHelper ph = new PrepareHelper();
    try{
      tm.prepare(Tx.getTransacData(), Tx.getId(), tc);
      ph.respAgree(txId);
    } catch(Exception e) { ph.respNotAgree(txId); }
    void triggerNext() {};
}
```

Fig. 11. 2PC invasive AWED aspect for the creation of the transactional behavior

perform nested calls in the prepare method using the `cflow` pointcut construct: this constructs forbids new replication actions within the dynamic extent of an open call to the `prepare` method.

Implementation using AWED. The result of the transformation[3] of the pattern program shown in Fig. 9 is a set of AWED aspects that implement the replication under pessimistic locking. Each aspect of the pattern-based solution is translated into an AWED aspect that modularizes source and target parts of a pattern expression. Figure 11 presents the resulting implementation of the `Aprepare` pattern-level aspect. In this case the generated source pointcut uses a sequence to explicitly relate the relevant transaction-related event to the call `send` that initiates replication, *i.e.*, farming out of the prepare action. The target advice executes the prepare method in the target caches and calls an `respAgree` or `respNotAgree` method to yield the answer.

5.2 Qualitative and Quantitative Evaluation

In Section 2 we have motivated that the current implementation of JBoss Cache is subject to problems concerning modularization, in particular, scattered and tangled code for the control of the transaction and replication concerns. Our solution improves the implementation in all those respects. First, each crosscutting

[3] We have applied the transformation manually for this evaluation but its automation is unproblematic.

concern is now modeled as an aspect and the choreography and interaction is defined without crosscutting by means of the pattern language (and AWED aspects on the implementation level). Second, distribution issues, coordination and composition of patterns are easily identifiable and modifiable in our solution. These advantages appear clearly in the `Aprepare` aspect: the source pointcut clearly defines the exact context (the sequence of method calls matched in the source pointcut) required to trigger the replication; furthermore, the related actions relevant to replication on different hosts are modularized in the aspect. Overall, our solution facilitates understanding and is easier to extend.

We have measured how our refactored version of JBoss Cache compares quantitatively to the plain JBoss Cache solution. For the corresponding experiments, we have considered transactions with pessimistic locking in JBoss cache. In the original code, there are more than 2674 LOC in 17 classes related to this concern. In our solution, the code consists of 532 LOC in 11 well-modularized aspects and classes: roughly a reduction of 80% of complexity (in terms of LOC). Most to this reduction is due to the fact that the transaction and communication protocol that is scattered and duplicated in `switch` structures is now re-factorized in well modularized entities.

6 Related Work

As to the best of our knowledge there is no directly related work that considers extensions to standard communication and computation patterns to accommodate crosscutting data accesses using AOP techniques. However, there are many approaches that are related in a weaker sense, in particular, approaches that use AOP for support for pattern implementations, sequential AOP systems that have been used with distributed infrastructures and more generally pattern-based approaches in distributed systems. We consider these groups of approaches in the following.

There have been several recent articles on support for the implementation of patterns using AOP. Hannemann and Kiczales in [16], in particular, show that several quality attributes, such as locality of definition and code reusability, of GoF pattern implementations can be improved through usage of AspectJ. Technically, these improvements are achieved by representing some roles in the pattern more concisely using AO abstractions. This is quite a different endeavor from ours that focuses on AOP as a support technology for the definition of an extended notion of patterns. However, the results on sequential pattern implementations using AOP should have analogues for distributed patterns and should be applicable to some extent to the invasive patterns we advocate.

A number of approaches have been put forward that use sequential AOP systems like AspectJ for the modularization of crosscutting functionalities in distributed and concurrent applications. These approaches — such as Eric Tanter's work on ReflexD [25], recent work on implementations of concurrency operators [11] and the approach of Concurrent Event-Based AOP [12] — while in principle be able to express invasive patterns as we have proposed, can only do so by modularizing crosscutting functionalities using separate aspects for each node in a distributed system. Our approach, through its pattern language but also on the implementation level through transformation into AWED, is much more declarative by directly expressing distribution-relevant relationships within single aspects, thus resulting in more concise programs that facilitate program understanding.

In the domain of distributed applications, several pattern catalogues have been proposed [7,20,2]. Such patterns are particularly widespread in component-based systems, *e.g.*, the CORBA and J2EE platforms [8,2]. These component systems provide communication and concurrency mechanisms that are used to implement patterns, *e.g.*, for the implementation of asynchronous broadcast services. However, these programming abstractions are not made explicit in the architectural description that defines the interconnection properties and, in contrast to our approach, no explicit means for the embedding of pattern-like interconnection structures in crosscutting contexts is provided.

In the more specific domain of (massively) parallel applications architectural and programming patterns are also quite popular. Much work has been done, for instance, on so-called skeletons following Cole's seminal work [9]. Recent work has focused on the application of such pattern-based parallelism to larger-scale imperative applications (see, *e.g.*, [24,21]). Most of these approaches essentially rely on an underlying regular communication topology and use of a homogeneous synchronization model, two properties that do not hold for the applications we are targeting. Furthermore, crosscutting accesses to execution state on which pattern applications are not addressed explicitly in such approaches.

Finally, several authors have proposed configurable frameworks to address the implementation of complex communication protocols by composition of simpler protocol entities (see, *e.g.*, [26]). However such approaches address protocol composition at a much lower level of abstraction (*e.g.*, TCP, UDP connections) than we consider.

7 Conclusion and Future Work

Software patterns have proven a versatile tool for program development. They facilitate application development and maintenance by raising the abstraction level of descriptions for software artifacts. Patterns have been very successful for sequential object-oriented applications, as well as for massively parallel algorithms. However, pattern-based approaches have been much less successful in the domain of distributed programming that are defined on irregular topologies and subject to inhomogeneous synchronization requirements. In this papers we have identified a major reason for the difficulty in applying programming patterns to distributed applications: applications of such patterns frequently depend on information that is not locally available where the pattern is to be applied.

In this paper we have proposed a solution: *invasive patterns*. Such patterns provide well-known computation and communication patterns (e.g., pipe, farm and gather) but also offer a built-in abstraction based on AOP for access to non-local state. We have motivated our approach in the context of JBoss Cache, a real-world infrastructure for transactional replicated caching. We have introduced a language for defining and composing *invasive patterns* that has been implemented by a translation into AWED, a system for explicitly distributed AOP. Finally, we have evaluated our approach qualitatively and quantitatively by presenting a non-trivial pattern-based refactoring of parts of JBoss Cache.

Our proposal provides a solid basis for numerous future work. First, *invasive patterns* currently support static only topologies, but AWED supports groups of hosts that evolve dynamically. Our language could easily be extended to benefit from this mechanism. Second, our semantics is a simple translation to AWED so it offers many optimization opportunities (e.g., aspects deployment on specific hosts, pattern composition specialization). Finally, patterns raise abstraction level of software and are prime candidates for formal methods (properties to be analyzed include communication protocol compliance, absence of deadlock, topology invariants, fault tolerance).

References

1. Akşit, M., Clarke, S., Elrad, T., Filman, R.E. (eds.): Aspect-Oriented Software Development. Addison-Wesley Professional, Reading (2004)
2. Alur, D., Malks, D., Crupi, J., Booch, G., Fowler, M.: Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies. Sun Microsystems Inc., Mountain View, CA (2003)
3. AOSD 2006. Proceedings of the 5th ACM Int. Conf. on Aspect-Oriented Software Development ACM Press (March 2006)
4. AspectJ home page, http://www.eclipse.org/aspectj
5. Awed home page, http://www.emn.fr/x-info/awed
6. Navarro, L.D.B., Südholt, M., et al.: Explicitly distributed AOP using AWED. In: AOSD 2006 (2006)
7. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. John Wiley and Sons Ltd., Chichester (1996)
8. Open Management Group (OMG). CORBA components, version 3
9. Cole, M.: Algorithmic Skeletons: Structured Management of Parallel Computation. MIT Press, Cambridge (1989)
10. IBM Corp. IBM Patterns for e-business Resources, http://www-128.ibm.com/developerworks/patterns/library
11. Cunha, C.A., Sobral, J.L., Monteiro, M.P.: Reusable aspect-oriented implementations of concurrency patterns and mechanisms. In: AOSD06 AOSD (2006)
12. Douence, R., Le Botlan, D., Noyé, J., Südholt, M.: Concurrent aspects. In: Proc. of GPCE 2006, ACM Press, New York (2006)
13. Douence, R., Fradet, P., Südholt, M.: A framework for the detection and resolution of aspect interactions. In: Batory, D., Consel, C., Taha, W. (eds.) GPCE 2002. LNCS, vol. 2487, pp. 173–188. Springer, Heidelberg (2002)
14. Easton, J., et al.: Patterns: Emerging Patterns for Enterprise Grids. IBM Redbooks (June 2006), http://publib-b.boulder.ibm.com/abstracts/sg246682.html
15. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Reading (1994)

16. Hannemann, J., Kiczales, G.: Design pattern implementation in java and aspectj. In: Proceedings of OOPSLA 2002, pp. 161–173. ACM Press, New York (2002)
17. JBoss Cache home page, http://labs.jboss.com/jbosscache
18. Kiczales, G.: Aspect oriented programming. In: Cointe, P. (ed.) ECOOP 1996. LNCS, vol. 1098, Springer, Heidelberg (1996)
19. Kiczales, G., Hilsdale, E., et al.: An overview of AspectJ. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, Springer, Heidelberg (2001)
20. Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects. John Wiley and Sons Ltd., Chichester (2000)
21. Siu, S., De Simone, M., Goswami, D., Singh, A.: Design patterns for parallel programming. In: Proc. of PDPTA 1996, vol. I, pp. 230–240. C.S.R.E.A. Press, University of Waterloo, Canada (1996)
22. Soares, S., Laureano, E., Borba, P.: Implementing distribution and persistence aspects with AspectJ. In: Norris, C., Fenwick Jr, J.B. (eds.) Proceedings of OOPSLA 2002, ACM SIGPLAN Notices, vol. 37(11), pp. 174–190. ACM Press, New York (2002)
23. Südholt, M.: Towards expressive, well-founded and correct Aspect-Oriented Programming. Habilitation thesis, University of Nantes (July 2007), http://www.emn.fr/sudholt/hdr/thesis.pdf
24. Tan, K., Szafron, D., et al.: Using generative design patterns to generate parallel code for a distributed memory environment. In: Proc. of PPOPP 2003 (June 2003)
25. Tanter, É., Toledo, R.: A versatile kernel for distributed AOP. In: Eliassen, F., Montresor, A. (eds.) DAIS 2006. LNCS, vol. 4025, Springer, Heidelberg (2006)
26. Wong, G.T., Hiltunen, M.A., Schlichting, R.D.: A configurable and extensible transport protocol. In: INFOCOM 2001. Proceedings of the 20th Annual Conference of IEEE Communications and Computer Societies, pp. 319–328. IEEE, Los Alamitos (2001)

NSLoadGen– A Testbed for Notification Services

Diego Palmisano and Mariano Cilia

Argentina Software Development Center - Intel Corp.
Cordoba, Argentina
firstname.lastname@intel.com

Abstract. During the past years a lot of work on Notification Services has been focused on features like scalability, transactions, persistence, routing algorithms, caching, mobility, etc. However, less work has been invested on how to evaluate or compare such systems. The selection of the most appropriate Notification Services for a particular application scenario is crucial and today available tools are bound to a particular implementation. If the Notification Service under test does not fulfill the application requirements then a new try with other Notification Service needs to be started from scratch: the description of the workload characterization and its injection cannot be reused.

In this paper we introduce **NSLoadGen** (Notification Services Load Generator), a testbed platform that supports the definition of real-life scenarios, the simulation of these scenarios against notification services, and finally generating vast data that can be used to precisely evaluate it. NSLoadGen is not targeted at any specific Notification Services, but rather is generic and adaptable. It has been designed to support a wide variety of Notification Services characteristics, hiding the many differences among messaging products/specifications (e.g. Java Message Service [1]) and, at the same time, it is easily extensible to support new implementations. This paper covers the different steps the tool follows (scenario definition, scenario simulation and result collection), the proposed approach, as well as relevant design and implementation details.

Keywords: Message-Oriented Middleware, Notification Services, Publish/Subscribe, JMS, Benchmarking, Distributed Testing, Simulation.

1 Introduction

Message-Oriented Middleware (MOM) refers to the software layer, between applications and the network protocols, that supports software engineers in developing distributed applications. Historically, MOM has been used to address issues related to heterogeneity, communication, and distribution of software components, relieving software engineers from the burden of solving low-level, network issues, such as lower-level communication protocols, concurrency control, transaction management, distributed object location, among others.

Notification Services are a kind of MOM that implement the publish/subscribe paradigm: a **message** is published by a **producer** in the Notification Services.

Consumers express their interest by issuing **subscriptions**. The Notification Services is responsible for the transportation of the published messages to the matching consumers. Normally, Notification Services platforms provide several QoS alternatives/dimensions such as reliability, security, delivery order, transactions, and so on. In spite of the availability of standardized solutions such as CORBA-NS (CORBA Notification Service) [2] or JMS (Java Message Service) [1], new kind of these infrastructures continue to be developed to address the needs of novel applications.

In the past years a lot of work has been focused on Notification Services features like scalability, transactions, persistence, routing algorithms, caching, mobility, etc. However, less work has been invested on how to evaluate or compare such systems. When surveying the existing implementations of standards (such as JMS) and publish/subscribe infrastructures, we found that they have several differences in the way they communicate with the client applications and the features they support. The selection of the most appropriate Notification Services for a particular application is crucial for its success. First of all, the workload characterization needs to be described and, afterwards, injected into the Notification Services under test. Unfortunately, available tools for this purpose are restricted in functionality and/or bound to a particular Notification Services. Consequently, if the Notification Services in question does not fulfill the application requirements, then a new try with another needs to be started from scratch.

In this paper we briefly introduce **NSLoadGen** (Notification Services Load Generator), a testbed platform that supports the definition of real-life scenarios for Notification Services, the simulation of these scenarios against notification services, and finally generating vast data that can be used to precisely evaluate it. NSLoadGen is not targeted at any specific Notification Services, but rather is generic and adaptable. It has been designed to support a wide variety of Notification Services characteristics, hiding the many differences among messaging products (even the many differences among JMS implementations) and, at the same time, it is easily extensible to support new Notification Services. NSLoadGen allows a high level definition of scenarios, independent of the Notification Services under test: a scenario can be described and conducted against different notification services. Moreover, it supports distributed simulation making it possible to assign injection of messages and subscriptions to different Notification Services where all the NSLoadGen instances will be coordinated among them.

The rest of the paper is organized as follows: related work is presented in **section 2**, focusing on Notification Services features relevant to NSLoadGen, benchmarking and testing of NSs. The proposed approach is described in more detail in **section 3**, presenting a brief description of the three steps supported by the platform. The main design decisions are sketched in **section 4**: how Notification Services independence was achieved, the architecture of producers and consumers and finally a description about the NSLoadGen distributed environment. Relevant implementation details are presented in **section 5**. **Section 6** presents conclusions, open issues and areas of future work.

2 Related Work

Since this work deals with various issues, like Notification Services platforms, distributed testbed, and benchmarking, this section serves as an introduction/overview of these topics including related work.

2.1 Notification Services

Our work relates to a subset of the features available in a Notification Services that are relevant in the context of load generation and injection. This features are:

– *Interface between Notification Services and Client Applications:* a Notification Services offers services to client applications by means of functions. At least it exposes two functions: (i) *subscribe* allowing consumers express their interest; (ii) *publish* allowing producers inject messages. Several Notification Services platforms extend this interface with additional functions called *advertise*, which a producer uses to inform the messages it will generate, *unsubscribe* and *unadvertise*. Subscriptions can be matched repeatedly until they are canceled by a call to unsubscribe. Advertisements remain in effect until they are canceled by a call to unadvertise.

– *Addressing Models:* Subscribers are usually interested in particular messages and not in all messages. The different ways of specifying the messages of interest have led to several distinct addressing models: *channel-based addressing* where the subscriber receives all messages that are posted to the channel; *subject-based addressing* [3] which is based on the notion of "topics" or "subjects" that represents a hierarchical organization of keywords to which participants can publish notifications and subscribe to; *content-based addressing* which allows consumers to express interest on the content of the messages; and *concept-based addressing* [4] that allows consumers in an heterogeneous environment to express their interest considering their local context thus delivering to them as a result *ready-to-process* data which does not require further conversions.

– *Architecture:* There are two major architectural alternatives:
  1. Centralized: The Notification Services has one point where the matching of messages with subscriptions is evaluated,
  2. Distributed: cooperating components (called brokers) evaluate the matching in a distributed way building a network of brokers.

  The centralized approach is simple to implement but it has a single point of failure and it could be a bottleneck. The distributed approach is more complicated to implement since a single global view on all subscriptions needs to be supported, but it scales much better.

Actually, there are several research projects involving Notification Services (e.g. REBECA [5], JEDI [6], SIENA [7], HERMES [8], GRYPHON [9], standard specifications (e.g. CORBA-NS [10], JMS [1]) and commercial products (e.g. TIB/Smartsockets and TIB/Rendezvous) that focus on different aspects of data dissemination, like efficient routing algorithms, optimal filter placement, minimization of resources usage, etc.

Examples of JMS providers are FiornaoMQ and SoniqMQ (commercials products) and ActiveMQ, OpenJMS, JBossMQ, and Presumo (open source projects).

2.2 Benchmarking of MOM

Benchmarking Notification Services is a process that involves two main goals: validation of the service interface, and performance evaluation [11]. The choice of applications, configurations, and workloads is clearly crucial for the formulation of the benchmark. Part of the benchmark must be an expression of real, current applications. In addition, configurations and workloads must reflect real computing environments and actual use scenarios. Under the same scenario, the test system could be executed over several notification services, allowing to know how well they perform under the scenario in question. The results can be used to compare performance of different Notification Services solutions, helping in the selection process of a messaging system.

Key areas in Notification Services benchmarking definitions are deployment topology (definition of distributed scenarios of consumers and producers), messaging domain (point-to-point or publish-subscribe systems[1]), duration, behavior of producers and consumers (how many messages, the distribution of injections), messages data and type, and so on. Furthermore, related to consumers, which messages they want to receive (i.e., subscription) and if they must be active at the moment of receiving messages are key points.

In the literature there are research projects in the field of load injection in Notification Services, which are mainly focusing on specific Notification Services rather than general. Further, they almost do not provide a way to describe real-life scenarios.

Sonic Software Test Harness [12] presents a simple test application that allows to create simple scenarios and test it over any JMS implementation. The behavior of TestHarness can be influenced by means of command-line parameters such as where is the JMS implementation running, number of producers and consumers to be created, number of connections to be used by producers and consumers, messages' length and test duration. Beside that, producer or consumer behavior can not be defined at all.

KOM ScenGen [13] relies on different steps in generating a scenario: topology creation, properties setting, network load or workload creation, plausibility check (where several things critical for the scenario can be checked for plausibility), scenario exportation to a collection of scripts and configuration files that are used to setup the scenario in a testbed, simulation and evaluation. Even thought it covers most of the necessary steps to follow a testing, the user has to have a deep understanding about the used technologies and settings.

[1] Point-to-point products are built around the concept of message queues.

Kuo and Palmer present a test harness that automates the testing of JMS implementations for correctness and performance [14]. The paper describes a methodology for black-box testing of a JMS provider. A test harness exercises the JMS server and a model of the expected behavior of this server is built; the model can then be compared against the actual behavior of the provider. However, like Sonic Software Test Harness, the user has not the possibility of setting producer/consumer behavior and workload characterization.

The idea behind Service Integration Test Tool (SITT) [15] is to test services and their workflow by analyzing the message flows. In this way, message injections and receptions are logged in a standardized manner. As the previous JMS test harness, SITT supports distributed testing and the execution is managed by a master node. Further, it allows to define scenarios by means of writing XML scripts. However, it falls in the same deficiencies: no description of producers and consumers behavior, no characterization of workload, lack of using data services to fill messages out.

The scenario descriptions of the tools mentioned above do not provide enough characteristics to be defined properly and, as a consequence, it is difficult to set up real-life behaviors. In addition, sometimes they can be defined with reduced power of expression, via command line arguments or directly hard coded in the application, having to recompile it every time a change is produced in the scenario. Besides that, producers are limited to simply inject messages as quick as possible with no chance to describe a pattern nor distribution of message injection. Messages are filled out with constant or even without data. Most commonly, consumers are restricted to subscribe to all messages and, as a consequence, the evaluation of the subscription functionality is left out.

Regarding the result presented during the test execution, most of the test tools generate a fixed number of metrics without allowing to create new ones. Even, the results are represented as plain numbers in the execution console. Consequently, the evaluation/comparison of Notification Services becomes very difficult.

3 Proposed Approach

This section presents the approach followed by NSLoadGen. The idea is to provide users the ability to (quickly and easily) express their specific scenarios and, as result, obtain information related to important events carried out during the simulation of the scenario. An experiment (a.k.a., test run) can be simply formulated as a three-phase process that is described in detail below.

3.1 Scenario Definition

NSLoadGen supports the definition of scenarios by means of a description language. The scenario definition and the platform functionality has been separated and, thereby, behavior of producers and consumers is user-defined and independent of the logic of the platform. To adjust the scenario, the user only has to modify the scenario description and not the code of the platform. The tool allows the definition of a wide range of properties related to producers (where and when to inject messages), consumers (where and when to inject subscriptions)

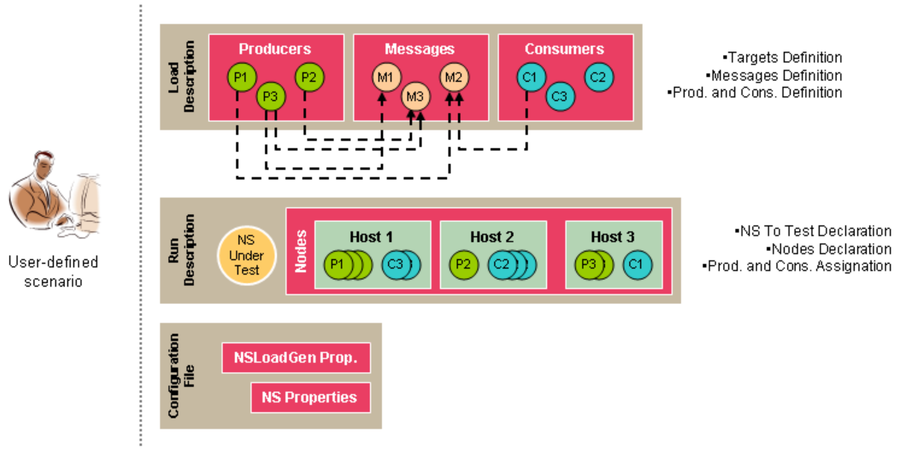


Fig. 1. Load Description, Run Description and Configuration File

and messages (types, data and sizes), and assignment of them to distributed nodes in a distributed environment. Therefore, the scenario definition was separated into three parts, each of them containing related data that, together, shapes a real-life scenario.

An example of the general structure of these three sections is depicted in figure 1. The first section defines *data and behavior*, containing producer, consumer and message definition, and how these three parts relate to each other. The second set up how producers and consumers are assigned to distributed nodes, i.e., *binding information*, and the last one is related to the *configuration* of the platform.

The idea behind dividing the scenario description in three parts is not only to make its definition clearer and easier, but it is also founded in the fact that normally only the data and behavior is modified during several test runs under the same binding setup. For instance, once the network location of producers/consumers is defined, they are likely to be maintained unchanged while messages structure and its assignment to producers and consumers may vary to evaluate different executions. Conversely, if the test goal is to use the same load but creating more producers and consumers along different tests, data and behavior information will be maintained unmodified and the binding part would be modified. With respect to the configuration part, it is usually stable across test runs. This section will change only if a different Notification Services is going to be tested.

3.2 Scenario Simulation

One of the most important requirements related to the simulation of the scenario is that it can be driven in a distributed environment. This is one of the characteristics that allows to create realistic scenarios for real-world application deployments. When multiple distributed producers and consumers are active at

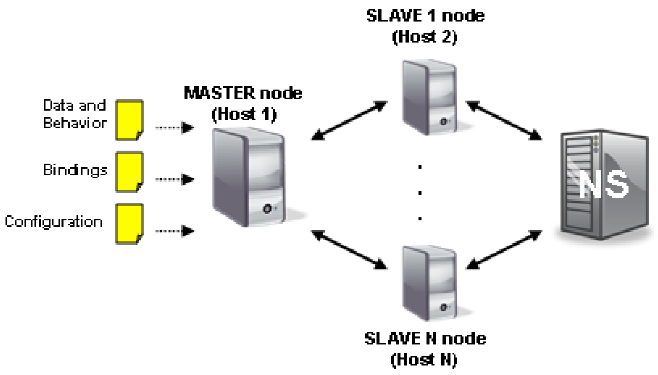


Fig. 2. Example deployment diagram

the same time, the notification service will be stressed from multiple directions, potentially causing the server to behave differently.

NSLoadGen allows to deploy a test across a number of distributed machines. According to the scenario description, a group of NSLoadGen instances are started in these computers, each of them simulating the part of the scenario that correspond to it. Figure 2 shows an example deployment diagram.

The experiment is coordinated by a *Master* node, responsible to start, stop and synchronize the set of NSLoadGen instances. Each NSLoadGen instance has assigned the instantiation of producers that inject messages (also advertisements/unadvertisements depending on the Notification Service) and/or the consumers that inject subscriptions (and unsubscriptions). These events are registered for later evaluation.

3.3 Result Collection

Commonly, users want to analyze different metrics of test runs, such as producer injection, message latency or even which was the injection distribution of a set of producers during a time windows. They may even want to have the same results using different charts. Thereby, instead of presenting specific metrics, NSLoadGen provides a very convenient way to catch events containing the most important actions carried out during the test run. This information can be used later to analyze in deep the behavior of the System Under Test (SUT).

4 Architecture and Design

This section sketches the principal architectural design presenting a brief description of the main building blocks.

4.1 High Level Architecture

The NSLoadGen platform is intended to be a testbed for notification services, maintaining it abstract enough to test different Notification Services. In this

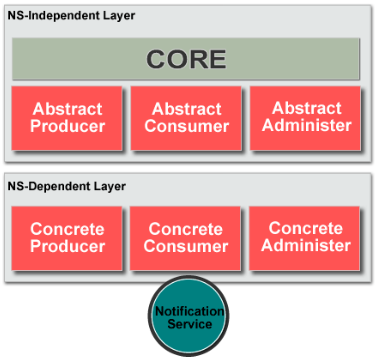


Fig. 3. Logical view of NSLoadGen

way, it can be seen as an independent piece of software that instantiates NS-independent (Notification Services independent) producers, consumers and inject messages and subscriptions. At the moment of interacting with the SUT, these pieces are transformed from NS-independent to NS-dependent (Notification Services dependent). Users only have to declare the scenario. NSLoadGen will simulate it and transform involved components (producers, consumers, messages, and so on) to communicate with the underlying Notification Services.

In this way, the architecture of NSLoadGen is logically split into two layers as can be seen in figure 3. The upper layer stays unaware of any concrete Notification Services functionality, while the lower one is in charge of interacting with the SUT. The idea was to encapsulate common functionality, needed for every Notification Services, to be used by the core of NSLoadGen, but letting specific functionality to be implemented by NS-dependent components. This is achieved by well-known pieces of software called **adapters**. The platform has specific interfaces defined as an integral part of its core. A library of adapters for several notification services is available in NSLoadGen. Additionally, users can write adapters for those Notification Services not supported yet.

At the producer side, **producer adapters** are responsible for issuing advertisements, unadvertisements and injecting messages. All of them are NS-independent activities. Only at the moment of carrying out some NS-dependent task, the platform will involve the respective dependent component. In the same way, NSLoadGen is also split into two parts at the consumer side; **consumer adapters** are in charge of making subscriptions, receiving messages and unsubscriptions. This component will call to specific consumers when necessary.

Additionally, there is a third kind of adapters called **administer adapter** and, as its name suggests, it has the responsibility of carrying out management activities like creating connections, message destinations[2] or queues.

Producer. Producers are mainly in charge of injecting messages according to an injection distribution specified by the user in the scenario description. As a

[2] A *Destination* is an abstract name used to mean “subject” or “topic”.

consequence, several steps have to be carried out to generate ready-to-send messages. The producer component design follows a pipeline architecture: *Message Factory* → *Message Filled Out* → *Message Transformer* → *Injection*. Producers have a behavior assigned, as defined in the scenario, that drives the message injection. In addition to this behavior, they have set up specific messages to inject. When the inject distribution demands the injection of a message several steps that are described later take place to get a ready-to-send message.

First, the *Message Factory* creates a new message according to the message structure and the message type to be injected (name-value pairs, a XML message or a serialized object). A single message structure can be linked with any of these types: a single producer can be assigned to inject the same message structure but with different types or even data. The *Message Factory* creates cleaned messages, that is, messages without data, and are NS-independent, that is, messages that are not specific to any notification service.

Created messages are the input to the *Message Filled Out* component. This component makes an analysis of the information the message should have, fetches the corresponded data from the source specified by the user in the scenario and fills the message out. Afterwards, a *Message Transformer* is needed to transform messages from NS-independent to NS-dependent in the third step. Fourth, the filled NS-dependent message is received by the *Producer Adapter* that finally, as it was previously explained in section 4.1, it has encapsulated common functionality needed for every Notification Services. This last component injects the messages into the Notification Services, delegating this task to the *NS Interface* component that contains functionality specific to the SUT.

Although this approach gains in Notification Services independence it has an important drawback: every time a message has to be injected a three-step process take place: message creation → fill it out with data → transformation. What is worse, these steps are executed after test run start up. As a consequence, it may become a bottleneck on NSLoadGen side and its performance can be seriously affected. In order to improve the performance of producer component, a *Pool* of ready-to-send messages was added to the previous solution. Figure 4 shows how the previous approach was adapted to support this pool. Note that messages are defined by means of a called *message structure* that is sketched with a tree.

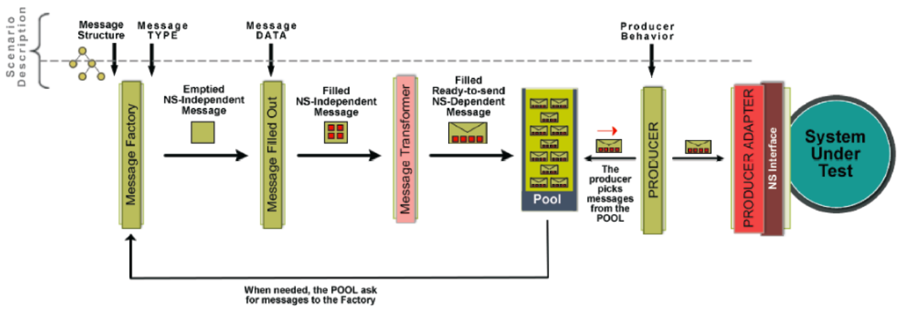

Scenario Description
Message Structure
Message TYPE
Message DATA
Producer Behavior
Message Factory
Emptied NS-Independent Message
Message Filled Out
Filled NS-Independent Message
Message Transformer
Filled Ready-to-send NS-Dependent Message
Pool
The producer picks messages from the POOL
PRODUCER
PRODUCER ADAPTER
NS Interface
System Under Test
When needed, the POOL ask for messages to the Factory

Fig. 4. Producer component: proposed approach

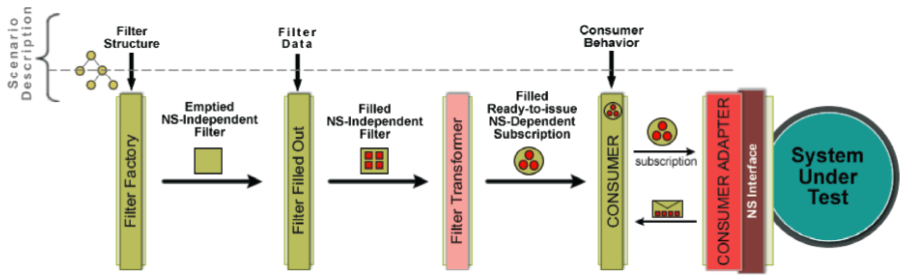


Fig. 5. Consumer component: proposed approach

As in the previous solutions, the producer component activates itself to inject a message. However, instead of starting the three-step sequence, it only picks a ready-to-send message from the pool. Once the pool returns a message, the producer gives it to the adapter that injects it into the SUT, always relying on the Notification Services Interface.

The pool acts as an active component that runs at background, when the system is not overloaded. A mark is used to maintain this pool with a minimal number of instances during runtime. When this mark is reached the pool will asks for new fresh message instances to the factory, triggering the three-step process. Notice that messages stored in the pool are ready-to-send. The producer only has to pick one and deliver it to the adapter. Additionally, NSLoadGen provides also configuration mechanisms to skip the step of filling out messages with data, becoming the three-step process in only two steps. This is very useful when message data is not important in the evaluation.

Consumer. Consumers are mainly in charge of issuing subscriptions and receiving messages. Each consumer has a filter[3] that was assigned by the user in the scenario. Like messages, filters go through a sequence of steps to convert it into subscriptions that the consumer can issue. Figure 5 shows the architecture of the consumer component.

The *Filter Factory* component creates a NS-independent filter according to the *Filter Structure*. The platform supports the dynamic creation of filter data, i.e., the user can express the filter structure and the source of data to fill it out, and NSLoadGen will create it. In this way, once the filter has been created it is moved to the *Filter Filled Out* component, which returns a filter with data, but still Notification Services independent. After that, the filter is transformed to a subscription and assigned to the consumer. During simulation and according to its specified behavior, a consumer will give the subscription to the *Consumer Adapter*, who relies on the *NS Interface* component to issue it. Later, the notification service will deliver messages to that consumer when matching messages are published.

Notice that filter steps are similar and different to message steps presented in the producer section. A filter has to traverse several components to become a

[3] A *filter* is an abstract name used to mean NS-independent subscription.

subscription. But in contrast to producers, this is not a cyclic process. Filters, and as a consequence subscriptions, are created once during the initialization of the test run. The same approach is used for advertisements on the producer side.

4.2 Distributed Testbed

NSLoadGen was designed to allow automated testing of notification services across a number of machines connected by a network. A test run is related to three primary concepts: the *System Under Test)*, one *Master* node and, commonly, several *Slave* instances. Test runs are coordinated by the Master node, who starts, stops and synchronizes the set of Slave instances. The *Slave* is in charge of instantiating producers that inject messages, and/or consumers that issue subscriptions. Its purpose is to wait for Master high-level commands and to trigger activities (start producers, consumers, collect data, etc.). The Master node has the ability to collect run-time data from all slave instances. This is done by simply sending a command after execution.

Figure 2 in section 3.2 shown an example deployment diagram of a test run. At the moment of starting the simulation, the input scenario (composed by the three parts presented in section 3.1) is received by the Master instance and sent to the Slave instances. Once it has been received, the Salve instance loads it and, after an initialization process where producers, consumers and the pool of messages are created, the Master is responsible for coordinating the Slave instances acting as initiator of the test run.

At this time, the Slave(s) begin processing their part of the scenario: producers will begin sending advertisements, messages and unadvertisements; and consumers will issue subscriptions, consume messages and unsubscriptions. As each message is sent or received, these events are registered with relevant information such as a unique message ID and timestamp[4]. Individual producers and consumers can be configured with different message production, persistence, durability, subscriptions and other characteristics, as well as connection and disconnection behavior, for every Slave instance.

At the end of the execution the Master node triggers the “stop” signal to the Slave instances, stopping producers and consumers.

4.3 Logging Approach

During the test run all significant events inside the platform are registered. This information can be obviously used to analyze the notification services under test, not only performance conclusions but also producer and consumer behavior, exception conditions, etc. The user has the possibility of defining which are the events that should be registered along the test run, as well as, the amount of information for every event (debug, info or minimal). Significant events are advertisements, unadvertisements and injections (in the producer side) and subscriptions, unsubscriptions and received messages (in the consumer side).

[4] In case of performance analysis want to be considered, NSLoadGen is dependent on all system clocks being synchronized.

However, it is clear that registering important events of an application is a well-known crosscutting concerns. We use aspect-oriented programming (AOP) to flexibly store relevant events during the experiment. Since the evaluation step is carried out relying on this data, supporting a configurable recording of it was very important. This approach allows maintaining independence between NSLoadGen and the tools to analyze the log. Aspects can be configured to manipulate the data and with different alternatives to store/disseminate it. For example, these events can be sent to files, or stored in a database, or sent them directly to a monitoring application that presents real-time results about the test run.

Furthermore, the user can decide which events to register. Depending on the metrics of interest, not all events are necessary. For example, if throughput is going to be analyzed, then only message injections and receptions should be caught, but not producer and consumer connections and disconnections. Also, the user can modify the format of the events. As an example, the user could define a XML format, a plain-string format, or SQL sentences, by means of extending the components that are in charge of event logging.

Additionally, the user can deactivate the logging module altogether if these events are not important. For example, Salvucci+ [16] rely on NSLoadGen as the load generation tool. This particular project focused on analyzing the internal of Notification Services at runtime where NSLoadGen logging features where disabled.

5 Implementation

NSLoadGen has been implemented in Java and also uses XSLT transformations. This section introduces the steps of *scenario definition* and *scenario emulation* in more detail, focusing on the implementation view.

5.1 Scenario Definition: Meta Language

As presented in section 3.1, a NSLoadGen scenario is composed by three parts. To describe it, a Meta Language in XML was defined where each scenario issue was assigned to a separated XML file: *Load Description* contains data and behavior, *Run Description* contains binding information and *Configuration* contains information to configure NSLoadGen and the SUT.

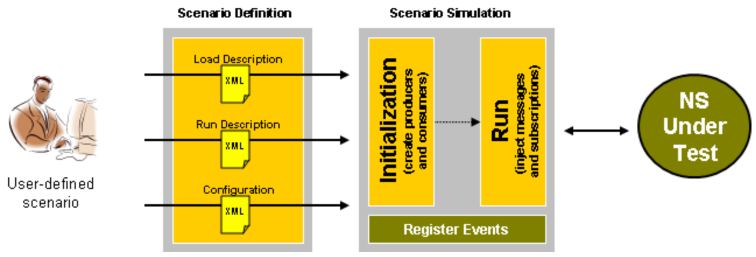


Fig. 6. Testbed steps

5.2 Scenario Simulation

In this step the defined scenario is simulated and the SUT is put under pressure. Along this step NSLoadGen passes through a two-phase process: *initialization time* where the platform is set to simulate the defined scenario, and *run time* where the simulation itself takes place.

Initialization Time. One of the main activities carried out in the initialization step is the conversion from {Message Structures,Message Type} to Message Factories. The *Load Description* part of the scenario contains information about the message structure, the source and data that the messages have to be filled out, and the message type that every producer should inject. Figure 7 illustrates the idea with an example.

In the example there are three producers, **P1**, **P2** and **P3** and three message structures declared. The message structures are declared as a tree, and, within every element of the tree there are information about how it has to be filled out with data (for example, node **c** must be a 20-length string). NSLoadGen allows different sources of data, such as from a data base, a custom source defined by the user, CVS files, or fixed data, to fill out a message. Furthermore, NSLoadGen has a library of functions that return common data used in different domains, such as city names, country names, zip code numbers, to name a few.

Additionally, the figure shows that producer **P1** is assigned to generate name-value pair messages according to the structure **M2**, producer **P2** is assigned to create XML messages following the structure **M3** and, finally, producer **P3** should generate messages **M1** and **M3** of types XML and serialized, respectively. By means of separating the message structure from the assignation of them to producers, the same kind of message can be transformed to different types. In the example, producer **P2** and **P3** should generate messages according to the structure **M3** but with type XML and serialized, respectively.

The activity of transforming a message structure with an assigned type to a real NS-independent message is carried out by specifics **Message Factory**s, as explained in subsection 4.1. Therefore, and following the previous example, the platform should create four Message Factory, one for each {message type,message structure} pair, such as "LoadGeneratorFactoryMessage_XML_Message1" and "LoadGeneratorFactoryMessage_NVP_Message2".

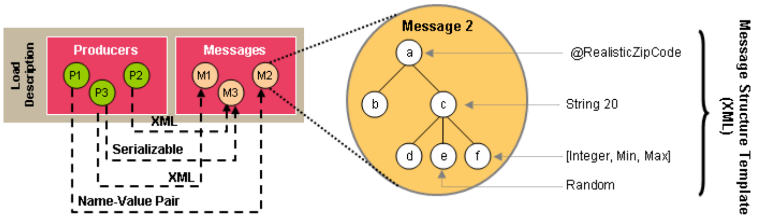


Fig. 7. Load declaration

Other activities that take place at initialization time are producers and consumers creation, message pool creation and subscriptions creation.

Run Time

Message Transformer. The system uses a generic message type along the test execution but, at specific times, it transforms this NS-Independent message to NS-Dependent message. This activity is carried out by a component called *Message Transformer*. In case a new notification service should be supported by NSLoadGen, the *MessageTransformer* interface has to be implemented. One of the pre-defined message transformer provided by NSLoadGen is `JMSMessageTransformer` which transform from NSLoadGen to JMS messages, and vice versa.

Message Types. NSLoadGen supports three types of messages: *XML*, *Map* and *Serializable* (See figure 8). These three types cover most of the message types used in Notification Services.

`LoadGeneratorMessage` is the message superclass and contains a set of properties. As was previously explained, these messages are created by factories. In this way, `LoadGeneratorFactoryMessage` is the superclass of the factories containing an abstract method *getMessage* that returns a `LoadGeneratoreMessage`.

Notification Service Independence. In order to fulfill with the requirement of Notification Services independence, NSLoadGen was desgined and implemented using the *Abstract Factory Design Pattern* [17]. This pattern allows to create families of related or dependent objects without specifying their concrete classes, which in turn, will be NS-dependent classes.

As can be seen in figure 9, the **AdapterAbstractFactory** provides the methods to create the adapter entities (e.g., **LoadGeneratorProducer**, **LoadGeneratorConsumer** or **LoadGeneratorAdminister**). This factory is created at initialization time and is NS-dependent, so it knows what kind (subclasses) of entities will be needed to interact with the SUT. It is in charge of creating NS-dependent entities. NSLoadGen interacts with these components without even knowing which Notification Services are they communicating with. In order to extend the platform to support new notification services, specializations of these classes should be written.

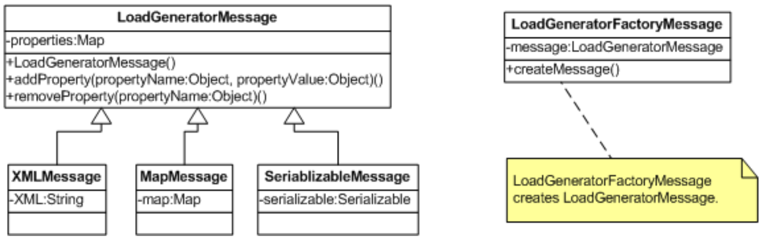

LoadGeneratorMessage
-properties:Map
+LoadGeneratorMessage()
+addProperty(propertyName:Object, propertyValue:Object)()
+removeProperty(propertyName:Object)()
XMLMessage
-XML:String
MapMessage
-map:Map
SeriablizableMessage
-serializable:Serializable
LoadGeneratorFactoryMessage
-message:LoadGeneratorMessage
+createMessage()
LoadGeneratorFactoryMessage creates LoadGeneratorMessage.

Fig. 8. Message class diagram

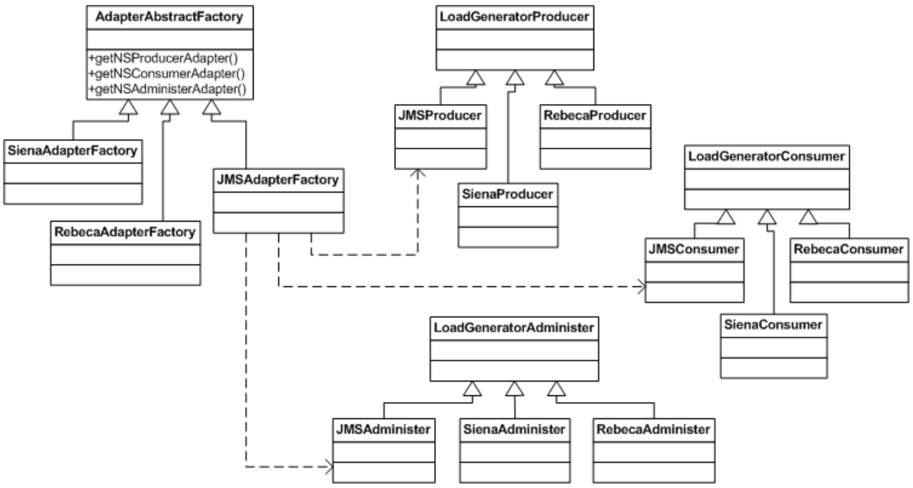


Fig. 9. Adapter abstract factory class diagram

Distributions to Simulate Behavior. Distributions are a way that the user can use to vary the producer and consumer behavior. At the producer side, they have the ability to connect to and disconnect from the system, or, in other words, they can issue an advertisement, inject messages for a while, and then disconnect from the system issuing an unadvertisement. As a consequence, they only can inject messages when their state is active. During this state, producers can emit messages following, probably different, inject distribution. For instance, let's suppose that a producer was assigned to a loop where it maintains enable during 20 seconds and disable for 10 seconds. In addition, when enable, it has an inject distribution that corresponds with the "sine" function.

To cope with this requirement, producer components have been designed with two clocks, one to drive the enable/disable function distribution, and the other to follow the injection rate. Obviously, the second clock only works when the first is active, so *enable/disable timer* is the "master" of the *injection timer*. Figure 10 (a) shows both timers and the events carried out by a producer, i.e., *enable*, *inject* and *disable*.

This important feature in producers of enabling and disabling (or advertisements and unadvertisements) is given by a distribution function that the user defines in the *Load Description* part of the scenario. A library of pre-defined functions are available but, additionally, users can define their own functions. Besides that, sometimes there are NSs that do not support advertisements and unadvertisements. In these cases, NSLoadGen has a default timer that wakes up the injection timer at the beginning of the test run, and sleeps it at the end. An important example of such systems is the JMS specification, which does not define the advertisement behavior.

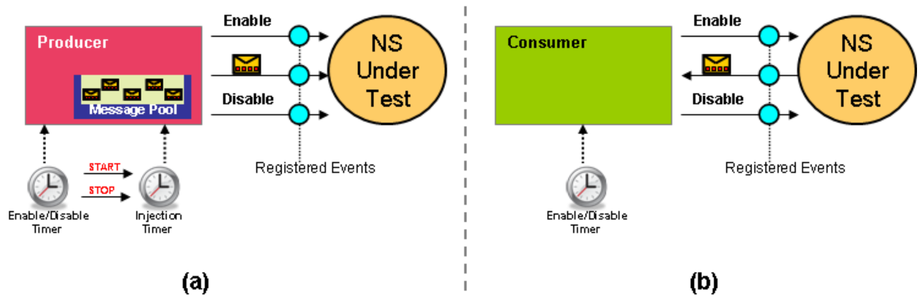


Fig. 10. Distribution timers: (a) producer side, (b) consumer Side

At the consumer side, among the main parameters is the rate of subscriptions and unsubscriptions. Unlike producers, consumers only have one timer associated that drives this rate. Figure 10 (b) shows the main concepts.

This enable/disable (subscription/unsubscription) clock drives the complete consumer's life-cycle. Like producers, this timer uses a enable/disable (subscribe/unsubscribe) distribution which is given by a function defined in the *Load Description* part. There are a library of pre-defined functions but, additionally, users can define their own function. In case the user does not want a subscription/unsubscription behavior for some consumer, the platform has a default timer that wakes up the consumer at the beginning of the test run, and sleeps it at the end.

6 Conclusions and Future Work

This paper presented NSLoadGen, a testbed for notification services that automates the process of stimulating a Notification Services. In other words, NSLoadGen is in charge of injecting load according to a scenario description with multiple purposes:

1. supporting the selection of the most appropriate Notification Services for a specific application (scenario);
2. analyzing a Notification Services under stress conditions for code optimization;
3. analyzing a Notification Services under typical and peak conditions for tuning purposes.

For doing so, we first concentrated on defining a language to describe the scenario to be simulated, covering as many Notification Services features as possible and demonstrating how that representation can be done easier by splitting it in three parts: one for the load characterization, one for running the experiments and one to configure the platform and Notification Services. After the scenario is specified, it is then taken and simulated in order to evaluate the characteristics of the Notification Services under test. Detailed data during simulation is generated for later analysis.

In this way, the platform covers scenario description, as well as load generation, deployment, execution and data collection during the experiment. NSLoadGen was constructed with the idea to be generic, meaning with this that it should be independent of the Notification Services under test. NSLoadGen has been successfully used with several JMS implementations and also with other NSs that do not provide a JMS interface [18]. Furthermore, the platform provides several facilities in which new messaging systems can be rapidly tested by only specializing a small set of classes.

The scenario description language was founded on XML . From the point of view of the end user, the interaction with the platform through the use of XML can be cumbersome, even more when complex scenarios have to be represented. Hence, in future versions a graphical editor will be provided to specify the scenario with a graphical (easy to use) interface generating as a result at the end the language already defined.

One interesting direction to be investigated is to extend NSLoadGen to stress Web Services and other SOA-related technologies. We are also analyzing the possibility to incorporate the notion of mobility of consumers and producers into the platform. Besides, studies on more complex scenarios are needed to validate the scenario description language, extending it if necessary.

References

[1] Inc. Sun Microsystems. The Java Message Service Specification. Technical report, Sun Microsystem Technical Report (2002)

[2] Object Management Group. Notification service specification (July 1999)

[3] Oki, B., Pfluegl, M., Siegel, A., Skeen, D.: The Information Bus - An Architecture for Extensible Distributed Systems. In: Proceedings of SIGOPS, pp. 58–68 (1993)

[4] Cilia, M., Antollini, M., Bornhövd, C., Buchmann, A.: Dealing with heterogeneous data in pub/sub systems: The Concept-Based approach. In: DEBS 2004,

[5] Mühl, G., Fiege, L.: The REBECA Notification Service (2001)

[6] Cugola, G., Di Nitto, E., Fuggetta, A.: Exploiting an event-based infrastructure to develop complex distributed systems. In: ICSE 1998

[7] Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems (2001)

[8] Pietzuch, P.R., Bacon, J.: Hermes: A distributed event-based middleware architecture. In: ICDCSW 2002

[9] Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R.E., Sturman, D.C.: An efficient multicast protocol for content-based publish-subscribe systems. In: ICDCS 1999

[10] Object Management Group. Common Object Request Broker Architecture specification

[11] Carzaniga, A., Wolf, A.L.: A benchmark suite for distributed publish/subscribe systems. Technical report, Department of Computer Science, University of Colorado (2002)

[12] Sonic Software. Benchmarking e-business messaging providers. Technical report, Sonic Software (2004)
[13] Heckmann, O., Pandit, K., Schmitt, J., Steinmetz, R.: KOM ScenGen - The Swiss Army Knife For Simulation And Emulation Experiments. In: Ventre, G., Canonico, R. (eds.) MIPS 2003. LNCS, vol. 2899, Springer, Heidelberg (2003)
[14] Kuo, D., Palmer, D.: Automated analysis of java message service providers (2001)
[15] Dustar, S., Haslinger, S.: Testing of Service Oriented Architecture - A practical approach (2004)
[16] Salvucci, S., Cilia, M., Buchmann, A.: A Practical Approach for Enabling Online Analysis of Event Streams. In: DEBS 2007
[17] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns, Elements of Reusable Object-Oriented Software. Addison Wesley Longman, Inc, Redwood City (1998)
[18] Palmisano, D., Cilia, M.: NSLoadGen, a testbed for Notification Services. In: partial fulfillment of the requirement for the degree of Systems Engineer (August 2006)

Ontologies, Databases and Applications of Semantics (ODBASE) 2007 International Conference

ODBASE 2007 PC Co-chairs' Message

As in recent years, the focus of the ODBASE conference lies in addressing research issues that bridge traditional boundaries between disciplines such as databases, artificial intelligence, networking, data extraction, or mobile computing. There has been an increasing focus on semantic technologies in ODBASE. The work on semantic modeling technologies is being progressively scaled up to handling millions of triples which permit adoption of semantic applications within a few days. The envelope is being progressively pushed out to enable even faster, wider and broader enterprise-wide and Web-scale applications. Also, ODBASE 2007 encouraged the submission of papers that examine the information needs of various applications, including electronic commerce, electronic government, mobile systems, or bioinformatics.

We had a very good response to the call for papers and 81 papers were submitted for ODBASE this year. The papers were rigorously reviewed and each of the papers had to have three peer reviews. The selection of papers was difficult because of the high quality of papers submitted. The final selection had 18 full and 4 short papers, and 10 posters. The accepted papers cover the themes of ontology mapping, semantic querying, ontology development, learning, text mining, annotation, metadata management and finally ontology applications.

For the papers in ontology mapping, Web semantics are used as background knowledge and a discovery methodology for semantic mappings between ontologies. Current implementations and future directions of ontology and database mappings are discussed and the interoperability of XML schema and OWL is also discussed. In addition methodologies for interpreting queries beyond their use for semantic interoperability which use SPARL++ for mapping RDF vocabularies, and language for retrieving ontology fragments are discussed in the semantic querying area. To assist with the development and evolution of the ontologies, papers are included which address a taxonomy construction methodology using similarity measures, an ontology for reactive rules, heuristics for Bayesian network-based geospatial ontologies, a pattern-based ontology construction approach, evolution of community-based knowledge intensive systems, and semantic Web methodology for managing image archives.

Learning methodologies for studying ontology mappings from human interactions and for search applications are also presented. Moreover, approaches for automatic feeding of knowledge bases using semantic representation of knowledge are demonstrated. Approaches for accessible browsing annotating unstructured metadata, matching of ontologies with XML schemas, and labeling of data extracted from the Web are discussed for metadata management.

Lastly, papers which consider various applications of ontologies for enhancing data quality of databases using ontologies, Web service-based annotation applications for marketing, categorization in eGovernment, and semantic matching in enterprises are presented.

These papers provided timely and stimulating discussions at the conference. We hope the readers find them stimulating, too.

August 2007

Tharam Dillon
Michele Missikoff
Steffen Staab

Towards Next Generation Value Networks

York Sure

SAP AG
york.sure@sap.com
http://www.sap.com

Abstract. We are moving towards a services economy where more and more value in an economy is created through services. A key enabler for such an economy is the transformation of services themselves into tradable goods similar to products. As organizations are focusing on core competences we will see (i) focused organizations will provide more specialized services (service providers), (ii) composition and coordination of services provided by different service providers into value-added services will become important business opportunity (service broker / coordinator), and (iii) organizations will be willing to "buy" more services from service providers and integrate them into their business operations (service consumers). This leads to longer and deeper service value chains consisting of a large number of services. A value chain may consist of services provided by a diversity of service providers, thus resulting in an increased complexity in coordinating services provided by multiple service providers.

The German funded TEXO project runs under the umbrella of the THESEUS research programme and addresses the challenges imposed by next generation value networks. The interdisciplinary TEXO consortium is coordinated by SAP and includes a number of partners having technical, economical and legal competencies. In this talk I will give examples of existing value networks, present the vision of the TEXO project for next generation value networks, present the interdisciplinary approach to address the technical, economical and legal challenges, and illustrate scenarios in which the TEXO solution adds value.

Speaker-Bio

Dr. York Sure is a Senior Researcher at SAP Research in Karlsruhe, working as Technical Coordinator for the Theseus/TEXO project. Previously he worked as Assistant Professor at the Institut AIFB of the Universität Karlsruhe (TH) in Germany where he lectured master and bachelor courses on Semantic Web and Computer Science. At the AIFB he was project leader for the EU IST FP6 Integrated Project SEKT, the EU IST FP6 Thematic Network of Excellence Knowledge Web where he was also appointed as research area manager, and the Vulan Inc. funded multi-stage international project Halo - Towards a Digital Aristotle (phase 2). After graduating in December 1999 in Industrial Engineering he received in May 2003 his PhD in Computer Science. From June to September 2006 York was appointed as a Visiting Assistant Professor at Stanford University. In 2006 he was awarded with the IBM UIMA Innovation Award and in 2007 with the doIT Software Award.

Combining the Semantic Web with the Web as Background Knowledge for Ontology Mapping

Ruben Vazquez and Nik Swoboda

Facultad de Informática, Universidad Politécnica de Madrid,
Campus de Montegancedo, 28660 Boadilla del Monte, Madrid, Spain
ruben.vazquez.sanchez@alumnos.upm.es,
nswoboda@clip.dia.fi.upm.es

Abstract. We combine the Semantic Web with the Web, as background knowledge, to provide a more balanced solution for Ontology Mapping. The Semantic Web can provide mappings that are missed by the Web, which can provide many more, but noisy, mappings. We present a combined technique that is based on variations of existing approaches. Our experimental results in two real-life thesauri are compared with previous work, and they reveal that a combined approach to Ontology Mapping can provide more balanced results in terms of precision, recall and confidence measure of mappings. We also discover that a reduced set of 3 appropriate Hearst patterns can eliminate noise in the list of discovered mappings, and thus techniques based exclusively in the Web can be improved. Finally, we also identify open questions derived from building a combined approach.

Keywords: Ontology Mapping, Background Knowledge, Semantic Web, Web.

1 Introduction

The feasibility of the Semantic Web, as the Web of automated-agents and web-services cooperating together[1], greatly relies on the idea of integrating a variety of ontologies [6]. This is a problem known as Ontology Mapping, which is seen by many as the *Achilles Heel* of the Semantic Web (van Harmelen in [21]).

Ontology Mapping is a basic operation not only in the Semantic Web but also in a number of different areas. A few examples of Ontology Mapping applications are Catalog Integration, P2P Databases, Agent Communication and Web Services Integration. As a result, a great deal of work in Ontology Mapping has been done recently. Some surveys appear in [10], [7], [13] and [19].

A definition for Ontology Mapping, or Ontology Matching, is the following. It takes as input two schemas/ontologies, each consisting of a set of discrete entities (e.g., tables, XML elements, classes, properties, rules, predicates), and

[1] See [5] for an original idea behind the Semantic Web, and [11] for a number of different meanings with which the Semantic Web is linked.

determines as output the relationships (e.g., equivalence, subsumption) holding between these entities [19].

Ontology Mapping can either be considered as a problem where basic structure (taxonomy) between terms exist, or as a task for automatizing the discovery of semantic links between flat, without hierarchy, lists of terms, being the later a case that has been little studied at the time of writing (see [19] for an in-depth survey of different views on Ontology Mapping). Therefore, if current term-matchers take, as input, lists of terms, in place of structured models, then they will probably fail. However, many ontologies are lightweight, i.e., they have a hierarchy with less than two or three levels, and thus structure can not be much exploited for Ontology Mapping [15]. Consequently, the complexity of the problem increases.

The problem we address in this paper is that of dealing with flat lists of terms taken as input to discover mappings among pairs belonging to different lists. Particularly, we limit our searches to subsumption relations ($\sqsubseteq$), since the gold standards we use only include this type of relation between terms.

We make four contributions. First, we design a framework that combines the Semantic Web with the Web for Ontology Mapping, test it, and analyze it in terms of precision, recall and confidence measure of mappings. The combined model is based on variations of techniques presented in [18] and [22]. Second, we compare our experiments with previous related work. Third, we find a small set of 3 Hearst patterns that yields better results for Ontology Mapping, in terms of precision, recall and confidence measure. By reducing a set of Hearst patterns from 6 to 3 appropriate ones, many wrong mappings can be left out, and many correct mappings can be uncovered. Fourth, we also point out open questions that require a great deal of effort in order to build a more solid combined approach for Ontology Mapping. Specifically, compound names, contradictions between derived mappings, quality of tools to search and the growth of both Webs in different domains are revealed as open questions.

This paper is organized as follows. Section 2 presents a brief state-of-the-art of solutions for Ontology Mapping based on Background Knowledge. Section 3 presents the data set used in the experiments, as well as the experiments. Sections 4 presents the architecture of the model we have designed. Section 5 provides results, jointly with an analysis and open questions. Conclusions are given in section 6.

2 Related Work

Many approaches to Ontology Mapping search directly for mappings between strings and structures. These approaches have a limitation: ontologies slightly different sintactically may be difficult to map. A less rigid solution is to use background knowledge for finding mappings [12], [18], [20], [22].

In [20], background knowledge is presented as an in-advance, manually, selected ontology that then acts as a bridge between input ontologies. The positive side of selecting ontologies manually is that a good choice, a complete and proper ontology for ontology mapping, can contribute to discovering mappings. On the other hand, a manual selection is too expensive in terms of time, and some applications do not allow previous selection of ontologies, instead, many applications in the Semantic Web need to find mappings automatically.

In [12] and [18], the Web is used as background knowledge by exploiting so-called *Hearst linguistic patterns* [1] (e.g., $\langle$concept-1$\rangle$ *such as* $\langle$concept-2$\rangle$) in order to discover relations between concepts. The weakness of this approach is that, in general, automatic knowledge extraction approaches lead to high levels of noise. Many mappings can be obtained, and thus finding both right and wrong links also increases. On the other hand, the strong point of using the Web as background knowledge for Ontology Mapping is that finding the right background knowledge does not need human intervention, but instead, it is done on the fly.

In [22], the Semantic Web is taken as background knowledge for Ontology Mapping. The advantage of this approach over manual selection methods is that ontologies are selected at run-time. The disadvantage of this approach is that it is totally dependent of the Semantic Web size, which is still highly limited in a number of different domains.

3 Experimental Set-Up

In this section we present the data set we used, and the experiments performed. The data set was provided by van Hage. The experiments were carried out with an application we developed in JAVA.

3.1 Dataset

Table 1 includes a short sample of 3 meat-food and 3 animal-meat mappings from gold standards. Particularly, the gold standards we used consist of 32 mappings from meat products to food types (meat-food), and 31 mappings from animal products to meat products (animal-meat).

Table 1. Sample of Gold Standards

Mappings	
meat-food	animal-meat
Ham $\sqsubseteq$ Foods	Pork $\sqsubseteq$ Meat Products
Chicken $\sqsubseteq$ Foods	Dove $\sqsubseteq$ Meat Products
Turkey Patties $\sqsubseteq$ Processed Foods	Turkey Patties $\sqsubseteq$ Minced Meat

Both the terms and the manually established mappings were obtained from the same dataset used in [18]. They compared a subset of two thesauri: the UN FAO's AGROVOC[2], and the USDA Nutrient Database for Standard Reference,

[2] http://www.fao.org/agrovoc

release 16 (SR-16)[3], particularly, two modules from AGROVOC (on food types with 21 concepts, and animal products with 88 concepts) against one module of USDA SR-16 (on meat products with 24 concepts).

3.2 Experiments

In running the experiments, pairs from meat-food and animal-meat lists are taken as input, and mappings are searched. In order to contrast our results with those from the gold standards, we limit our searches to subsumption relations ($\sqsubseteq$), since the gold standards only include this kind of relations between pairs of terms. We also calculate a confidence measure for each mapping, following the work in [16] and [17], where a *mapping element* is defined as a 5-tuple consisting of: id, entity one, entity two, confidence measure, and a relation.

Using the Web. Table 2 shows a series of experiments we ran taking the Web as background knowledge. We followed the technique described by Cimiano and Staab in [12], and then replicated by van Hage, Katrenko and Schreiber in [18], as one of their four ontology-mapping techniques. This method is described in Section 4. In van Hage et al's technique 6 Hearst patterns were used. In contrast to van Hage, Katrenko and Schreiber's method, we ran experiments by using 6 Hearst patterns, as well as with only 3 Hearst patterns.

Table 2. Experiments Using the Web

Experiment	Data set	# Hearst patterns
1	meat-food	6
2	meat-food	3
3	animal-meat	6
4	animal-meat	3

Another difference with van Hage, Katrenko and Schreiber's technique was that they used the Google API. By contrast, we use the Yahoo API, which makes it harder to compare, but sheds some light on differences between Yahoo and Google APIs. The reason why we decided to use the Yahoo API is that the Google API keys needed to run the experiments were no longer issued as of December 5, 2006[4].

In our experiments, we searched subsumption relations both ways for each pair of terms. For example, given the pair of terms (*Bacon*,*Pork*), we searched both $Bacon \sqsubseteq Pork$ and $Bacon \sqsupseteq Pork$.

Using the Semantic Web. Table 3 shows a series of experiments we ran with the Semantic Web as background knowledge. We followed variations of two strategies proposed by Sabou, d'Aquin and Motta in [22], so-called S1 and S3. S1

[3] http://www.nal.usda.gov/fnic/foodcomp/Data/SR16/sr16.html

[4] http://code.google.com/apis/soapsearch/reference.html

is a method based on one single ontology to discover mappings. S2 is a technique performing cross-ontology mapping searches. We used Swoogle API as they did too. Further details and examples on both strategies are given in Section 4.

Table 3. Experiments Using the Semantic Web

Experiment	Data set	Strategies
1	meat-food	S1
2	meat-food	S1, S3
3	animal-meat	S1
4	animal-meat	S1, S3

Combining the Webs. We designed an extremely simple algorithm to combine results derived from both Webs, and infer mappings. In particular, we designed an in-parallel approach, in which Web-based and Semantic Web-based matchers were run independently, and then results were combined. Results of the combined approach were derived manually. Details on this algorithm are provided in the next section.

3.3 Measures

Note that in our experiments we used three measures to analyze the quality of the results. Particularly, we take into account precision, recall, and confidence measure of mappings. We describe how confidence measure is defined in Section 4. Mapping precision and mapping recall are defined as follows:

$$Mapping\ Precision = \frac{|correct\ mappings\ discovered|}{|total\ mappings\ discovered|} \qquad (1)$$

$$Mapping\ Recall = \frac{|correct\ mappings\ discovered|}{|mappings\ in\ reference\ Golden\ Standards|} \qquad (2)$$

4 Architecture of the Model

In this section, we justify the architecture of our combined approach, describe the two elemental models that we take as starting building blocks, as well as variations we have introduced on them, and present the architecture of the combined approach.

4.1 Motivation

We built a combined approach on the assumption that each Web can contribute its own advantages, and the hypothesis that an effect of positive balance may emerge when merging the Web and the Semantic Web as sources to discover mappings. Consequently, we expected to create a new technique able to exhibit advantages from each single method, as well as able to avoid disadvantages, in terms of precision, recall and confidence measure of mappings at the same time, specifically:

- assumption, the Web can find many more mappings than the Semantic Web, it is better in terms of recall;
- assumption, the Semantic Web can discover mappings with a maximum degree of confidence measure (100%), and precision may be very high too;
- hypothesis, an appropriate combined approach may increase the number of discovered mappings (recall) of the Semantic Web based approach, make the precision of the Web-based approach higher, and provide mappings with higher confidence than those obtained from the Web-based technique.

4.2 Using the Web as Background Knowledge

Algorithm 1 represents an iterative process that ends after considering all pairs of terms. For each pair of terms from meat-food or animal-meat, depending on what list of terms is being processed, the following steps are repeated:

1. Initialize to zero the number of hits for each subsumption relation.
2. For each Hearst pattern, send queries to Yahoo to add number of hits for each subsumption relation.
3. Draw a subsumption relation between the pair of terms, based on number of subsumption relations hits.

Table 4 shows the set of 6 Hearst patterns, and the set of 3 Hearst patterns we considered to run the experiments. Note that patterns marked with * were included in the set of 3 Hearst Patterns (H3).

Algorithm 1. Using the Web for OM with Hearst Patterns

```
Input: $LP$ List of pairs $p_i$; $LHP$ List of Hearst patterns $hp_j$
Output: $LSR$ List of subsumption relations $sr_i$
 1: for all $p_i$ in $LP$ do
 2:   $n_\sqsubseteq := 0$
 3:   $n_\sqsupseteq := 0$
 4:   for all $hp_j$ in $LHP$ do
 5:     $n_\sqsubseteq := n_\sqsubseteq$ + #hits for Yahoo query $(p_i, hp_j, \sqsubseteq)$
 6:     $n_\sqsupseteq := n_\sqsupseteq$ + #hits for Yahoo query $(p_i, hp_j, \sqsupseteq)$
 7:   end
 8:   if $n_\sqsubseteq > n_\sqsupseteq$ then
 9:     $sr_i := \sqsubseteq$
10:   else if $n_\sqsubseteq < n_\sqsupseteq$ then
11:     $sr_i := \sqsupseteq$
12:   else
13:     $sr_i := null$
14:   fi
15: end
```

Table 4. Hearst patterns used in the experiments

	⟨concept-1⟩	*such as*	⟨concept-2⟩	*
	⟨concept-1⟩	*including*	⟨concept-2⟩	*
	⟨concept-1⟩	*especially*	⟨concept-2⟩	*
such	⟨concept-1⟩	*as*	⟨concept-2⟩	
	⟨concept-1⟩	*and other*	⟨concept-2⟩	
	⟨concept-1⟩	*or other*	⟨concept-2⟩	

Below are two particular examples of results obtained with the approach based on the Web, with both 6 and 3 Hearst patterns.

Example 1. Given the pair (*beef*, *foods*), with 6 Hearst patterns, 5637 hits are returned to derive *beef* $\sqsubseteq$ *foods*, and 914 hits are returned to derive that *beef* $\sqsupseteq$ *foods*. Consequently, our technique concluded that *beef* $\sqsubseteq$ *foods* was the correct mapping, with a confidence measure of 86%, (5637 / (5637 + 914)) x 100.

Example 2. Given the pair (*beef*, *foods*), with 3 Hearst patterns, 5468 hits are for drawing *beef* $\sqsubseteq$ *foods*, and 0 hits are for concluding *beef* $\sqsupseteq$ *foods*. Our techniques concluded that *beef* $\sqsubseteq$ *foods* was the correct mapping, with a confidence measure of 100%, (5468 / (5468 + 0)) x 100.

Note that to decide what mapping (or subsumption relation) to derive, $\sqsubseteq$ or $\sqsupseteq$, our method simply selected the one with a higher number of hits. For example, given the pair of terms (*beef*, *foods*), with 6 Hearst patterns, the number of hits to derive *beef* $\sqsubseteq$ *foods* was 5637, and the number of hits to derive *beef* $\sqsupseteq$ *foods* was 914, then our technique derived *beef* $\sqsubseteq$ *foods*, because 5637 is greater than 914. This is how we dealt with contradictions in the Web-based technique: by selecting the subsumption relation with a higher number of hits.

4.3 Using the Semantic Web as Background Knowledge

The algorithm grounded on the Semantic Web is simple: for each pair of terms, ontologies containing them are searched by using Swoogle, and for each of those ontologies, subsumption relations are searched.

Algorithm 2. Using the SW for OM

```
Input: $LP$ List of pairs $p_i$; recursivity $rec$ ($true$, $false$)
Output: $LSR$ List of subsumption relations $sr_i$
1:  $n_{maxOnto} := 10$
2:  for all $p_i$ in $LP$ do
3:    $LOnto$ := Swoogle query ($p_i$, $n_{maxOnto}$)
4:    while $LOnto$ not empty and $sr_i$ not found do
5:      $sr_i$ := search subsumption relation ($p_i$, $LOnto_j$, $rec$)
6:    end
7:  end
```

More formally, the following steps are executed:

1. Initialize the maximum number of Ontologies to retrieve, $n_{maxOnto}$. We ran experiments with $n_{maxOnto} = 10$, retrieving 10 ontologies, at most.
2. For each pair of terms, obtain $n_{maxOnto}$ ontologies at most, by querying Swoogle.
3. For each pair of terms and each ontology, search subsumption relations, until one of them is found, or no more ontologies left.

This algorithm is inspired by Sabou et al's *Strategy One* (S1). The idea of this strategy is to find mappings within one single ontology, given a pair of terms. To do that, given a term t_1, a list of superclasses is obtained. Then, it is checked whether the other term t_2 belongs to the list of superclasses. If so, the mapping $t_1 \sqsubseteq t_2$ is derived. If no mappings are found in this search, a simmilar search is peformed for subclasses of term t_1, and a parallel reasoning is carried out to infer the mapping $t_1 \sqsupseteq t_2$.

Note also that we have included the recursivity parameter *rec*, which allows varying S1, and performing a cross-ontology mapping discovery. The idea behind recursive cross-ontology search is the following. If a mapping cannot be discovered in a single ontology, more than one ontology can be considered to discover a mapping between a pair concepts. For example, if we are searching for the subsumption relation $cat \sqsubseteq animal$, we could find: 1, $cat \sqsubseteq mammal$ in ontology O_{C-M}; 2, $mammal \sqsubseteq animal$ in ontology O_{M-A}. Then, we could draw the mapping $cat \sqsubseteq animal$ by considering the two ontologies O_{C-M} and O_{M-A}. This so-called *Strategy Three* (S3) by Sabou et al gave us inspiration to build our cross-ontology approach.

Sabout et al also proposed a *Strategy Two* (S2), extending Swoogle's coverage by making more flexible the process of finding names. In particular, they propose to perform: 1, string normalization; 2, dealing with compound names; 3, exploiting semantic relations between terms. The work we present is focused on developing techniques that exploit metadata in the Semantic Web, rather than dealing with terminological algorithms. Therefore, we left out an implementation of S2 in our approaches.

Two different strategies can be distinguished at this point to find mappings: S1 and S3, which involve two different algorithms, so-called Algorithm 3 and Algorithm 4 respectively. Algoritm 3 searches for mappings in one single ontology. Algorithm 4 is a recursive version that discovers mappings grounded in several ontologies.

Algorithm 3 searches for superclasses of t_1 within one ontology, and if term t_2 is included in that list, then it is concluded that $t_1 \sqsubseteq t_2$. If the search of superclasses fails, a new search of subclasses is started for t_1, and if the term t_2 is included in the list of subclasses, it is derived that $t_1 \sqsupseteq t_2$.

Algorithm 4 searches for superclasses of t_1 within one ontology, and starts recursively Algorithm 2 with parameters: $Lsuper_j$, t_2, $false$. Consequently, a mapping between $Lsuper_j$ and t_2 is searched in one sole ontology, stopping recursivity ($rec = false$). If a subsumption relation $Lsuper_j \sqsubseteq t_2$ is discovered, then it is derived the mapping $t_1 \sqsubseteq t_2$. If the search of superclasses fails, a new

Algorithm 3. Searching for subsumption relations in one ontology ($rec = false$)

Input: Pair p_i with terms t_1, t_2; O Ontology
Output: Subsumption relation sr_i

```
1:  if $sr_i$ not found then
2:    $Lsuper$ : = search superclasses ($t_1$, $O$)
3:    if $t_2$ in $Lsuper$ then $sr_i := t_1 \sqsubseteq t_2$
4:  fi
5:  if $sr_i$ not found then
6:    $Lsub$ : = search subclasses ($t_1$, $O$)
7:    if $t_2$ in $Lsub$ then $sr_i := t_1 \sqsupseteq t_2$
8:  fi
```

search of subclasses is started for t_1, and if it is discovered $Lsuper_j \sqsupseteq t_2$, then it is derived the mapping $t_1 \sqsupseteq t_2$.

Note that we only introduced one level of recursivity in our experiments. However, cross-ontology searches can be more complex, including higher levels of recursivity. Finally, we actually ran a variation of Algorithm 4, in which superclasses and subclasses of t_2 were searched too, and then followed a reasoning parallel to that of t_1 to derive mappings.

Algorithm 4. Searching for subsumption relations in several ontologies ($rec = true$)

Input: Pair p_i with terms t_1, t_2; O Ontology
Output: Subsumption relation sr_i

```
1:  if $sr_i$ not found then $Lsuper$ : = search superclasses ($t_1$, $O$)
2:  while $Lsuper$ not empty and $sr_i$ not found do
3:    temp-sr := Algorithm 2 ($Lsuper_j$, $t_2$, $false$)
4:    if temp-sr == ($Lsuper_j \sqsubseteq t_2$) then $sr_i := t_1 \sqsubseteq t_2$
5:  end
6:  if $sr_i$ not found then $Lsub$ : = search subclasses ($t_1$, $O$)
7:  while $Lsub$ not empty and $sr_i$ not found do
8:    temp-sr := Algorithm 2 ($Lsub_j$, $t_2$, $false$)
9:    if temp-sr == ($Lsub_j \sqsupseteq t_2$) then $sr_i := t_1 \sqsupseteq t_2$
10: end
```

4.4 Combining the Semantic Web with the Web as Background Knowledge

State-of-the-art ontology-mapping systems are not made of a single elementary matcher, but of combinations proposed to provide a more robust solution [19]. A possible classification of combined techniques is the following: 1, in-sequence elementary matchers, called hybrid matchers in [4], with examples available in [2] and [3]; 2, in-parallel matchers, called composite matchers in [4], which combine the results, e.g., taking average or maximum values from indepedently executed matchers (see [8], [9] and [14]). In this paper, we build an in-parallel approach to combine results obtained from both Webs.

Algorithm 5. Combining the SW with the Web for OM

```
Input: $LP$ List of pairs $p_i$; recursivity $rec$
Output: $LMC$ List of pairs ($map_i$, $conf_i$)
1:  $mappings\text{-}web$ := Algorithm 1 ($p_i$)
2:  $mappings\text{-}sw$ := Algorithm 2 ($p_i$, $rec$)
3:  for all $p_i$ in $LP$ do
4:    if $mappings\text{-}web_i$ not null and $mappings\text{-}sw_i$ not null then
5:      if $mappings\text{-}web_i == mappings\text{-}sw_i$ then
6:        $map_i := mapping\text{-}web_i$
7:        $conf_i := 100\%$
8:      fi
9:    else if $mappings\text{-}web_i$ not null and $mappings\text{-}sw_i$ null then
10:     $map_i := mappings\text{-}web_i$
11:     $conf_i := confidence(p_i)$
12:   else if $mappings\text{-}web_i$ null and $mappings\text{-}sw_i$ not null then
13:     $map_i := mappings\text{-}sw_i$
14:     $conf_i := 100\%$
15:   else
16:     $map_i := null$
17:     $conf_i := null$
18:   fi
19: end
```

We have designed an extremely simple algorithm grounded on the Semantic Web and the Web to infer mappings. Its main steps are the following:

1. Get mappings for pairs of terms grounded in both the Semantic Web and the Web
2. For each pair, if both Webs have found mappings, and they are not contradictory, derive that mapping with 100% confidence.
3. For each pair, if only the Web returns a mapping, derive it with a degree of confidence based on the number of hits of the mapping.
4. For each pair, if only the Semantic Web has discovered a mapping, derive that mapping with a maximum degree of condifence, 100%.

Note that contradictions between both Webs may appear. We did not have to deal with contradictions in our experiments. However, contradictions have to be tackled to provide a more robust solution.

5 Results

In this section we present the results of our experiments and analyze them. This section is divided into four sub-sections: 5.1, combining the semantic web with the web; 5.2, comparing Hearst patterns; 5.3, comparing new experiments with previous work; 5.4, open questions.

Table 5. Mappings grounded on both single techniques and a combined approach

Data set	Method	Precision	Recall	Confidence
meat-food	Yahoo (H3)	95% (18/19)	56% (18/32)	99%
meat-food	Swoogle (S1,S3)	100% (7/7)	21% (7/32)	100%
meat-food	Combined approach	95% (18/19)	56% (18/32)	100%
animal-meat	Yahoo (H3)	79% (15/19)	48% (15/31)	79%
animal-meat	Swoogle (S1,S3)	100% (6/6)	19% (6/31)	100%
animal-meat	Combined approach	81% (17/21)	55% (17/31)	81%

5.1 Combining the Semantic Web with the Web

Table 5 shows that mapping results grounded on both Webs differ in terms of precision, recall and confidecene measure. The Semantic Web provides the best results in terms of precision because all of the mappings found at the Semantic Web in our experiments are correct. The fact that our technique stops after finding the first mapping probably avoids adding noise, or more wrong mappings. On the other hand, the Web-based technique is also able to provide high precision when the appropriate Hearst patterns are selected. A further discussion on how to increase precision by using particular Hearst patterns is given in Section 5.2. The combined approach benefits from two new mappings found in the Semantic Web ($dove \sqsubseteq meat$, and $duck \sqsubseteq meat$) in order to increase the precision of the Web-based approach, in the animal-meat data set (see Table 5). By contrast, when the combined approach is applied to the meat-food data set, all of the mappings found at the Semantic Web have already been found by the Web-based technique. As a result, precision degree of the combined approach remains the same as that of the Web-based method, in the meat-food data set (see Table 5).

The Web provides better results in terms of recall than the Semantic Web for Ontology Mapping. Many more mappings can be drawn when using the Web currently, and Hearst patterns provide a good way to exploit this fact. The Web can be seen as the *big brother* of the Semantic Web, in the sense that it was created around 1990 while the Semantic Web is still taking off. A practical consequence of that is the amount of available data in each Web, the Web having much data from which to extract relations between pairs of terms, and the Semantic Web still lacking. As a result, the number of mappings found in the Web is simply much higher.

The combined approach takes advantage of considering all of the mappings discovered from both sources, the Web and the Semantic Web. Therefore, in terms of recall, the combined technique always provides equal or better results than the best single method. In particular, the combined approach equals the level of recall (56%) provided by the Web-based technique when applied over the meat-food data set (see Table 5); and, the recall degree of the Web-based technique (48%) is exceeded by the combined approach (55%) in the animal-meat data set (see Table 5), because two mappings that are discovered by the Semantic Web, and therefore included in the results given by the combined approach, are missed by the technique grounded only on the Web.

The confidence measure of mappings discovered in the Semantic Web is always the maximum, 100%, by our definition. We conceded this top degree based on the assumption that these mappings are built on purpose, which is an important distinction with the technique based on the Web where mappings emerge automatically by counting number of correct hits, out of total ones, returned by searches with Hearst patterns. The Web-based technique provides a quite high level of confidence for mappings, because many *wrong hits* (in the sense that those mappings are not correct according to the gold standards) returned by the Web are removed when considering only 3 particular Hearst patterns.

The combined approach increases the average confidence measure of mappings with respect to the Web-based technique. The combined technique gets the maximum confidence measure for every mapping, so that it is expected to maintain or increase the average confidence measure from the Web-based technique. For example, the mapping $poultry \sqsubseteq food$ has a 99% confidence according to the Web approach, however, since this mapping is also discovered by the Semantic Web approach, it gets a 100% confidence, i.e., the maximum between both confidence degrees ($max\{99\%, 100\%\}$), in the combined approach. In this way, the average confidence measure of the combined approach increases up to a 100% level in the meat-food data set (see Table 5).

5.2 Comparing Hearst Patterns

Table 6 shows that three Hearst patterns clearly yield much better results in terms of precision, as well as an increase in the level of recall, and a decrease of the number of wrong mappings, in both data sets. To explain this, we show how many hits returned each Hearst pattern in each data set on average, for both correct ($\sqsubseteq$) and wrong ($\sqsupseteq$) mappings (see Table 7).

Table 6. Mappings with 6 and 3 Hearst patterns

Data set	Method	Precision	Recall	# Wrong mappings
meat-food	Yahoo (H6)	56% (17/30)	53% (17/32)	1
meat-food	Yahoo (H3)	95% (18/19)	56% (18/32)	0
animal-meat	Yahoo (H6)	29% (10/35)	32% (10/31)	10
animal-meat	Yahoo (H3)	79% (15/19)	48% (15/31)	1

Table 7 shows that two Hearst mappings are clearly not good for Ontology Mapping: $\langle$c-1$\rangle$ *and other* $\langle$c-2$\rangle$, and $\langle$c-1$\rangle$ *or other* $\langle$c-2$\rangle$. These two Hearst patterns do not return many desirable (correct) mappings on average, e.g., $\langle$c-1$\rangle$ *or other* $\langle$c-2$\rangle$ provides 0 correct mappings on the meat-food data set, and also, they discover many undesirable mappings, e.g., $\langle$c-1$\rangle$ *or other* $\langle$c-2$\rangle$ finds 115 wrong mappings on average in the meat-food data set.

Table 7 shows that the remaining 4 Hearst patterns may be good enough for discovering mappings. However, a more detailed analysis exhibits the following particular feature on this set of patterns. $\langle$c-1$\rangle$ *such as* $\langle$c-2$\rangle$, and $\langle$c-1$\rangle$ *including* $\langle$c-2$\rangle$ are both clearly good candidates to include within a definitive set of

Table 7. Average # of hits for correct ($\sqsubseteq$) and wrong ($\sqsupseteq$) mappings

Data set	Hearst Pattern	# Correct hits	# Wrong hits
meat-food	$\langle$c-1$\rangle$ *such as* $\langle$c-2$\rangle$	680.03	0.00
meat-food	$\langle$c-1$\rangle$ *including* $\langle$c-2$\rangle$	161.40	0.46
meat-food	$\langle$c-1$\rangle$ *especially* $\langle$c-2$\rangle$	19.46	0.00
meat-food	*such* $\langle$c-1$\rangle$ *as* $\langle$c-2$\rangle$	27.18	0.18
meat-food	$\langle$c-1$\rangle$ *and other* $\langle$c-2$\rangle$	1.46	115.00
meat-food	$\langle$c-1$\rangle$ *or other* $\langle$c-2$\rangle$	0.00	19.40
animal-meat	$\langle$c-1$\rangle$ *such as* $\langle$c-2$\rangle$	383.93	0.06
animal-meat	$\langle$c-1$\rangle$ *including* $\langle$c-2$\rangle$	67.19	0.38
animal-meat	$\langle$c-1$\rangle$ *especially* $\langle$c-2$\rangle$	7.45	0.00
animal-meat	*such* $\langle$c-1$\rangle$ *as* $\langle$c-2$\rangle$	4.09	0.96
animal-meat	$\langle$c-1$\rangle$ *and other* $\langle$c-2$\rangle$	1.00	72.67
animal-meat	$\langle$c-1$\rangle$ *or other* $\langle$c-2$\rangle$	0.03	12.96

Hearst patterns, since the difference between the number of correct and wrong mappings is tremendous. With respect to the two remainder Hearst patterns, $\langle$c-1$\rangle$ *especially* $\langle$c-2$\rangle$, and *such* $\langle$c-1$\rangle$ *as* $\langle$c-2$\rangle$, the difference between number of correct and and wrong mappings is not that significant, and a particular detail leads us to include $\langle$c-1$\rangle$ *especially* $\langle$c-2$\rangle$ in our final set of best Hearst patterns, and to leave out the other pattern. Specifically, the main difference between these two patterns is that one of them does not draw wrong mappings in our experiments, and the other one does derive wrong mappings. This is the reason we included $\langle$c-1$\rangle$ *especially* $\langle$c-2$\rangle$, and did not include the other one.

Going back to the analysis of Table 6, we have clearly shown in Table 7 that there exists a direct relation betwen what Hearst patterns are selected for Ontology Mapping and the number of correct/wrong hits. Therefore, a good choice of Hearst patterns may decrease number of wrong mappings, and may provide both a higher recall, and a higher precision, since noise is removed, i.e., wrong mappings are left out, and correct mappings are uncovered.

5.3 Comparing New Experiments with Previous Work

Table 8 shows a significant difference between the Google and Yahoo APIs in terms of precision. If 6 Hearst patterns are used, the number of wrong mappings is decreased by the Yahoo API with respect to the Google one. As a result, the level of precision is higher in Yahoo.

In the meat-food data set, Google's precision is 30%, which is lower than Yahoo's precision, 56%. In the animal-meat data set, Google's precision, 17%, is again lower with respect to that of Yahoo's 29%. Note that recall degree is the same for both APIs in each different data set, 53% in meat-food and 32% in animal-meat.

Table 9 shows levels of recall for Sabou et al's and Vazquez's methods to be dissimilar from each other. To explain that difference, we can observe where those discovered mappings come from. In meat-food data set, Sabou et al's method

Table 8. Mappings discovered with the Web

Data set	Method	Precision	Recall	Source
meat-food	Google (H6)	30% (17/56)	53% (17/32)	van Hage et al
meat-food	Yahoo (H6)	56% (17/30)	53% (17/32)	Vazquez
meat-food	Yahoo (H3)	95% (18/19)	56% (18/32)	Vazquez
animal-meat	Google (H6)	17% (10/58)	32% (10/31)	van Hage et al
animal-meat	Yahoo (H6)	29% (10/35)	32% (10/31)	Vazquez
animal-meat	Yahoo (H3)	79% (15/19)	48% (15/31)	Vazquez

Table 9. Mappings discovered with the Semantic Web

Data Set	Method	Precision	Recall	Source
meat-food	Swoogle (S1,S3)	100% (8/8)	25% (8/32)	Sabou et al
meat-food	Swoogle (S1,S3)	100% (7/7)	21% (7/32)	Vazquez
meat-food	Swoogle (S1,S2,S3)	100% (11/11)	34% (11/32)	Sabou et al
animal-meat	Swoogle (S1,S3)	100% (1/1)	3% (1/31)	Sabou et al
animal-meat	Swoogle (S1,S3)	100% (6/6)	19% (6/31)	Vazquez
animal-meat	Swoogle (S1,S2,S3)	100% (4/4)	12% (4/31)	Sabou et al

mappings have been found in three ontologies, while results from Vazquez's method come from two ontologies (see Table 10). In animal-meat data set, Sabou et al's technique only discovered one ontology, while Vazquez's method found mappings in three different ontologies (see Table 10). Note that both series of experiments were run at different points in time (i.e., variations in lists of top ontologies returned for a given pair of terms could be introduced by Swoogle), and with different implementations (i.e., different maximum numbers of ontologies in which to search for mappings).

As pointed out by Sabou et al, results also show that techniques based on the Semantic Web may enhance results in terms of precision (and confidence of mappings), but cannot be used in isolation, because recall is still too low (see Table 9). On the other hand, the Web-based technique provides precisely what the Semantic Web based approach lacks of: a high recall. Therefore, a complementary approach, based on several techniques seems to be a good way to follow when searching for a more balanced method in overall terms.

5.4 Open Questions

Compound Names. Table 11 shows that precision is high without considering compound names, in both data sets. It also exhibits a number of irregular results regarding precision with compound names, and recall regardless of compound

[5] http://morpheus.cs.umbc.edu/aks1/ontosem.owl
[6] http://139.91.183.30:9090/RDF/VRP/Examples/tap.rdf
[7] http://www.ksl.stanford.edu/projects/DAML/UNSPSC.daml
[8] http://reliant.teknowledge.com/DAML/Mid-level-ontology.daml
[9] http://reliant.teknowledge.com/DAML/SUMO.owl

Table 10. Mappings discovered with the Semantic Web

Data set	Strategy	Sabou et al	Vazquez
meat-food	S1	$Beef^6, Pork^6 \sqsubseteq Food$ $Poultry^6 \sqsubseteq Food$	$Beef^5, Pork^6 \sqsubseteq Food$ $Ham^5, Poultry^6 \sqsubseteq Food$
meat-food	S3	$Ham^9, Duck^{8,6}, Goose^{8,6} \sqsubseteq Food$ $Turkey^{8,6}, Chicken^{8,6} \sqsubseteq Food$	$Chicken^{5,6}, Duck^5 \sqsubseteq Food$ $Ostrich^5 \sqsubseteq Food$
animal-meat	S1	$Bacon^6 \sqsubseteq Pork$	$Bacon^6, Beef^5 \sqsubseteq Pork$ $Ham^5, Chicken^7 \sqsubseteq Meat$ $Dove^7, Duck^7 \sqsubseteq Meat$

Table 11. Mappings with and without compound names

Data set	Method	Precision	Recall	Compound
meat-food	Yahoo (H3)	93% (14/15)	100% (14/14)	excluded
meat-food	Yahoo (H3)	100% (4/4)	22% (4/18)	only
animal-meat	Yahoo (H3)	79% (11/14)	44% (11/25)	excluded
animal-meat	Yahoo (H3)	57% (4/7)	66% (4/6)	only

names. Precision with compound names is variable depending on the data set (100% in meat-food, and 57% in animal-meat). Recall with compound names is also variable (22% in meat-food, and 66% in animal-meat); the same is true of recall without compound names (100% in meat-food, and 44% in animal-meat). Consequently, we consider that more experiments are needed to draw conclusions regarding compound names and Web-based techniques. We also note that the Semantic Web based technique did not return mappings between compound names. Therefore, compound names in the Semantic Web are particularly hard to be dealt with currently.

Contradictions. In order to decide what subsumption relation to draw, our Web-based techniques select the subsumption relation with a higher number of hits. In contrast, the Semantic Web technique stops after discovering the first mapping, given a pair of terms. In this way, no contradictions may appear in the Semantic Web based method. However, if the Semantic Web based technique does not stop searching for mappings after the first one is discovered, contradictory mappings may arise. In this case, an strategy able to deal with contradicitions in the approach based on the Semantic Web has still to be developed. It remains an open question. Sabou et al have already shed some light on how to tackle it. They consider that: 1, contradictory mappings can coexist in a contextualized form, i.e., by connecting mappings to the ontologies that they belong to; 2, why not to rely only on context-similar ontologies to derive mappings? In this way, contradictions are likely to disappear by considering only right ontologies, given a particular context; in this case, more advanced techniques to choose ontologies become crucial, and Swoogle's features (or those from other Semantic Web search engines) play a crucial role in this issue.

We did not have to deal with contradictions between Web-based and Semantic Web-based mappings in our experiments. However, we envisage it as a potential line of work, for building a more robust algorithm, to automatically provide coherent answers when contradictions between both Webs appear. One potential solution, extending ideas proposed by Sabou et al and applying them to both Webs, involves finding contextualized mappings in both the Web and the Semantic Web, and consequently contradictions would be likely to not appear.

Quality of tools to search and Growth of the Webs. If Yahoo, Google, Swoogle, or other search engines that may arise are able to provide more sophisticated options for automatic search, such as searches based on context, dealing with contradictions can become much easier as pointed out above. In this way, we find a direct relation between quality of mappings discovered, in terms of precision, and capability of search tools to do searches based on contexts. The confidence measure of mappings is likely to benefit too from contextualized searches, in the sense that many wrong mappings may disappear, and thus quality of mappings may increase.

Another interesting aspect to remark is the the growth of the Webs in many varied domains, which is also likely to provide more data that can be used to find more mappings, thus increasing the level of recall.

6 Conclusions

In this paper, we have designed, tested and analyzed a framework for Ontology Mapping based on the idea of combining the Semantic Web with the Web as background knowledge. We have performed experiments on real-life datasets and have compared our results with previous related work. We discovered that Web-based techniques that use Hearst patterns can provide better results, in terms of precision, recall and confidence measure of mappings, by selecting three appropriate Hearst patterns. We have also identified as open questions that require more study: performing experiments and analyzing results with compound names, dealing with contradictions between derived mappings, improving quality of tools to search, and growing of both Webs in different domains.

Finally, we have provided evidence to support our initial hypothesis that a combined approach may exhibit a more balanced solution in general terms. Particularly the combined approach benefits from advantages provided by partial techniques, and avoids disadvantages in terms of precision, recall and confidence measure of mappings.

Acknowledgments. Thanks to Marta Sabou for her comments; and to Willem van Hage for providing his experimental data set.

References

1. Hearst, M.: Automatic Acquisition of Hyponyms from Large Text Corpora. Proceedings of the 14th International Conference on Computational Linguistics (1992)
2. Bergamaschi, S., Castano, S., Vincini, M.: Semantic integration of semistructured and structured data sources. SIGMOD Record 28(1), 54–59 (1999)
3. Madhavan, J., Bernstein, P., Rahm, E.: Generic schema matching with Cupid. In: VLDB. Proceedings of the Very Large Data Bases Conference, pp. 49–58 (2001)
4. Rahm, E., Bernstein, P.: A survey of approaches to automatic schema matching. The International Journal on Very Large Data Bases (VLDB) 10(4), 334–350 (2001)
5. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (2001)
6. Hendler, J.A.: Agents and the Semantic Web. IEEE Intelligent Systems, 30–37 (March/April 2001)
7. Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-Based Integration of Information A Survey of Existing Approaches. In: IJCAI 2001. Workshop: Ontologies and Information Sharing (2001)
8. Do, H.H., Rahm, E.: COMA - a system for flexible combination of schema matching approaches. In: VLDB. Proceedings of the Very Large Data Bases Conference, pp. 610–621 (2001)
9. Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Learning to map ontologies on the semantic web. In: WWW. Proceedings of the International World Wide Web Conference, pp. 662–673 (2003)
10. Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: the state of the art. The Knowledge Engineering Review 18(1), 1–31 (2003)
11. Marshall, C.C., Shipman, F.M.: Which semantic web? In: HYPERTEXT 2003. Proceedings of the fourteenth ACM conference on Hypertext and hypermedia, pp. 57–66. ACM Press, New York (2003)
12. Cimiano, P., Staab, S.: Learning by Googling. SIGKDD Explor. Newsl. 6(2), 24–33 (2004)
13. Noy, N.F.: Semantic integration: a survey of ontology-based approaches. In: SIGMOD Record, ACM Press, New York (2004)
14. Ehrig, M., Sure, Y.: Ontology mapping - an integrated approach. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 76–91. Springer, Heidelberg (2004)
15. Schlobach, S.: Semantic clarification by pinpointing. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, Springer, Heidelberg (2005)
16. Bouquet, P., Euzenat, J., Franconi, E., Serafini, L., Stamou, G., Tessaris, S.: D2.2.1: Specification of a common framework for characterizing alignment. Technical report, NoE Knowledge Web project delivable, (2004) http://knowledgeweb.semanticweb.org/
17. Euzenat, J.: An API for ontology alignment. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 698–712. Springer, Heidelberg (2004)
18. van Hage, W., Katrenko, S., Schreiber, G.: A Method to Combine Linguistic Ontology-Mapping Techniques. In: Proc. of ISWC (2005)
19. Shvaiko, P., Euzenat, J.: A Survey of Schema-based Matching Approaches. Journal on Data Semantics IV (2005)

20. Aleksovski, Z., Klein, M., ten Katen, W., van Harmelen, F.: Matching Unstructured Vocabularies using a Background Ontology. In: Staab, S., Svátek, V. (eds.) EKAW 2006. LNCS (LNAI), vol. 4248, Springer, Heidelberg (2006)
21. van Harmelen, F.: Semantic Web Research anno 2006: main streams, popular fallacies, current status and future challenges. In: Klusch, M., Rovatsos, M., Payne, T.R. (eds.) CIA 2006. LNCS (LNAI), vol. 4149, Springer, Heidelberg (2006)
22. Sabou, M., dAquin, M., Motta, E.: Using the Semantic Web as Background Knowledge for Ontology Mapping. In: Sabou, M. (ed.) Proceedings International Workshop on Ontology Matching (OM-2006), collocated with ISWC 2006 (2006)

Discovering Executable Semantic Mappings Between Ontologies

Han Qin, Dejing Dou, and Paea LePendu

Computer and Information Science
University of Oregon
Eugene, OR 97403, USA
{qinhan, dou, paea}@cs.uoregon.edu

Abstract. Creating *executable* semantic mappings is an important task for ontology-based information integration. Although it is argued that mapping tools may require interaction from humans (domain experts) for best accuracy, in general, automatic ontology mapping is an AI-Complete problem. Finding matchings (correspondences) between the concepts of two ontologies is the first step towards solving this problem but matchings are normally not directly executable for data exchange or query translation. This paper presents an systematic approach to combining ontology matching, object reconciliation and multi-relational data mining to find the executable mapping rules in a highly automatic manner. Our approach starts from an iterative process to search the matchings and do object reconciliation for the ontologies with data instances. Then the result of this iterative process is used for mining *frequent queries*. Finally the semantic mapping rules can be generated from the frequent queries. The results show our approach is highly automatic without losing much accuracy compared with human-specified mappings.

1 Introduction

The emergence of the Semantic Web has emphasized the need for systems that are able to query, integrate and exploit data from multiple, disparate sources which may use different ontologies, but are about the same domain. Research involving the Semantic Web is experiencing huge gains in standardization in that OWL becomes the W3C standard for ontological definitions in web documents. However, it is extremely unreasonable to expect that ontologies used for similar domains will be few in number [5]. As the amount of data collected in the fields of Biology and Medicine grows at an amazing rate, it has become increasingly important to model and integrate the data with ontologies that are biologically meaningful and that facilitate its computational analysis. Hence, efforts such as the *Gene Ontology* (*GO*) [16] in Biology and the *Unified Medical Language System* (*UMLS*) [19] in Medicine are being developed and have become fundamental to researchers working in those domains. However, different labs or organizations may still use different ontologies to describe their data.

Some ontology-based information integration systems have been developed to process queries and exchange data from data resources with different ontologies.

A survey can be found in [32]. For example, the InfoSleuth [4] project provides an agent and ontology based infrastructure for information gathering and analysis in distributed environments such as the Web. In OBSERVER [23] the information sources are described by domain ontologies, and user queries are rewritten by using the inter-ontology matchings. In our own previous work, we used OntoEngine (an inference engine) and our expressive Web-PDDL ontology language to translate Semantic Web documents and answer queries [13] between ontologies. We enhanced these tools with the ability to handle relational databases in addition to Semantic Web documents in OntoGrate [11,12].

Ontology matching and mapping is the key step to enable ontology-based information integration. The goal of ontology matching is to generate correspondences between the concepts from different but related ontologies. In most cases, formal mapping rules with clear semantics need to be generated for information integration systems. The problem of finding matchings and mappings between information resources has been extensively studied by different research communities. To be clear, we distinguish between *matching* (correspondence) and *mapping*. Matching pairs related concepts. One matching example is "both property *phone* and property *work_at* in one ontology are matched to property *workphone* in another." However, mapping not only pairs concepts but also formally defines the relationships between them. For example, "for all Office which has a *phone* number, all People who *work_at* that office must have *workphone* as the same number" is a mapping between property *phone*, *work_at* and *workphone*.

Research in discovering and representing semantic mappings is still in very preliminary stages. Indeed, the ideal choice of the mapping language, one carefully balancing expressivity with scalability, is an open question. Current mapping languages, such as Datalog, F-Logic, DLR, KIF or LOOM, are more or less subsets of first order logic [31]. In this paper, we mainly use general first order logic syntax to represent mapping rules, such as the above mapping we mentioned:

$$\begin{aligned}&\forall x, y, z, People(x) \land Office(y) \land String(z)\\ &\qquad \land work\_at(x, y) \land phone(y, z) \rightarrow workphone(x, z)\end{aligned}$$

Our OntoEngine takes these kinds of rules in Web-PDDL[1] for information integration. The rules also can be represented as Datalog, SWRL [1] or other logic languages which some tools can process. Therefore, we call them *executable* mappings. It is similar as that Clio [24,17] calls *operational* mappings for database schemas.

In general, automatic ontology mapping is an AI-Complete problem. Many challenges remain. For example, although it is argued that mapping tools may require interaction from domain experts for best accuracy [29], it is not clear what kind of interaction between the system and domain experts should be supported. In this paper, we propose to combine ontology matching, object

[1] Web-PDDL is also a subset of first order logic. We call mapping rules in Web-PDDL syntax bridging axioms in our previous work [13,11,12].

reconciliation and multi-relational data mining to find the executable mapping rules as automatically as possible. Our approach starts from an iterative process to search the matchings and do object reconciliation for the ontologies with data instances. Then the result of this iterative process could be used for mining *frequent queries*. Finally the semantic mapping rules can be generated from the frequent queries. The results show our approach is highly automatic without losing much accuracy compared with human-specified mappings.

The rest of the paper is organized as follows. We first give some background and related work in Section 2. Then we demonstrate our general framework for ontology mapping in Section 3. We elaborate our iterative approach for ontology matching and object reconciliation in Section 4. In Section 5, we show our extension to a well-known multi-relational data mining tool (i.e., FARMER [25]) for supporting mapping rules discovery. We show some promising results by case studies with real ontologies in Section 6. We conclude the paper and discuss the future directions in Section 7.

2 Related Work

In this section we first give background and more detail of some existing schema or ontology matching and mapping systems. We also introduce an object reconciliation system and several Multi-Relational Data Mining systems.

Not surprisingly, the database community was one of the first to invest considerable effort in developing systems that match different database schemas (see [29] for a survey). Most schema matching systems, such as LSD [8], CUPID [20], iMap [7] and COMA [14], focus on retrieving correspondences between attributes using a variety of similarity or correlation heuristics. Similarly, research in knowledge engineering and the Semantic Web has resulted in tools for ontology matching that are absolutely critical in ontology-based information integration. These tools also show promising applications in the database arena (see [26] for a survey). Chimaera [22], Protégé [27], GLUE [9] and MAFRA [21] are some examples of such systems. GLUE [9] employs machine learning and exploits data instances to find matchings between concepts. It uses domain knowledge and domain-independent constraints to increase matching accuracy. Chimaera [22] provides a ontology editor to allow user to merge ontologies. It suggests potential matchings based on the names of properties but needs users to verify them. The disadvantage of this system is that it leaves what to do entirely to users. Protégé [27] gives initial ontology alignments by plugging in one of existing matching algorithms and focuses on guiding users to refine alignments. This system updates its suggestions based on the input of user and gives new alignments to user. MAFRA [21] is an interactive, incremental and dynamic framework, which builds a semantic bridging ontology for distributed ontologies. These semantic bridges specify how to translate entities from source ontology to target ontology. BMO [18] can generate block matchings using a hierarchical bipartition algorithm. This system builds a virtual document for each ontology and compares each pair of concepts with the information in the virtual document. This algorithm is efficient but it does not figure out how two blocks are matched, however these matching blocks still can be very helpful for finding mappings.

Clio [24,17] is a schema mapping system that can generate operational (i.e., executable) mappings in different formats such as SQL and XQuery for database integration. This system uses semantic information to find matchings first and allows the user to modify them or add new matchings. Then it produces mapping rules based on matchings. It is semi-automatic since it needs human interactions. The research by An, Borgida and Mylopoulos [3,2] provides very interesting methods to construct complex mapping rules between relational tables or XML data and ontologies when given an initial set of correspondences between the concepts in the schemas and ontologies. They offer the mapping formalisms to capture the semantics of XML or relational schemas by constructing the semantic trees from them. Their generated rules will be useful to domain experts for further refinement, as well as to applications. Our approach is to combine ontology matching, object reconciliation and multi-relational data mining to find executable mapping rules in a highly automatic manner.

The object reconciliation problem is studied for determining whether two different objects of data sets refer to the same real-world entity. Xing Dong et al [10] propose to reconcile the object references in three steps. The first step is constructing the dependency graph, which describes the relationships between object pairs. One object pair (i.e., a reconciliation decision) needs to be decided as reconciled or not. The next step is an iteratively re-computing process, which computes the similarity scores of reconciliation decisions. Since the similarity score of one reconciliation decision can both affect and be affected by the similarity score of its neighbors, the algorithm sets a fixed point to stop the process. Finally, transitive closure is computed for the final reconciliation results.

Multi-relational mining techniques are already applied in many research areas such as subgraph mining. FOIL [28] is a first-order learning system which can generate Horn rules for target predicates by examining both positive and negative examples. WARMER [6] is designed for frequent pattern mining in relational databases. This system is built on the foundations of Inductive Logic Programming (ILP) and does not need negative examples. FARMER [25] is also a frequent pattern mining system, it takes object identity as the starting point and introduces several optimizations, and thus it has better performance than WARMER. In this paper, we will borrow some ideas from FARMER for generating ontology mapping rules. We will give more detail of our algorithm in Section 5.

3 Framework

In this section, we first introduce our general framework for mapping discovery, then we illustrate our idea with some simple examples. Given a source ontology and a target ontology which model the same domain, ontology matching can find that some of their concepts (e.g., classes and properties in OWL ontologies) are matched to each other. Object reconciliation can find that some instances from both ontologies represent the same real world entities. Our final goal is to generate executable mapping rules based on that information.

3.1 System Architecture

Figure 1 shows the architecture of our system. There are five main components.

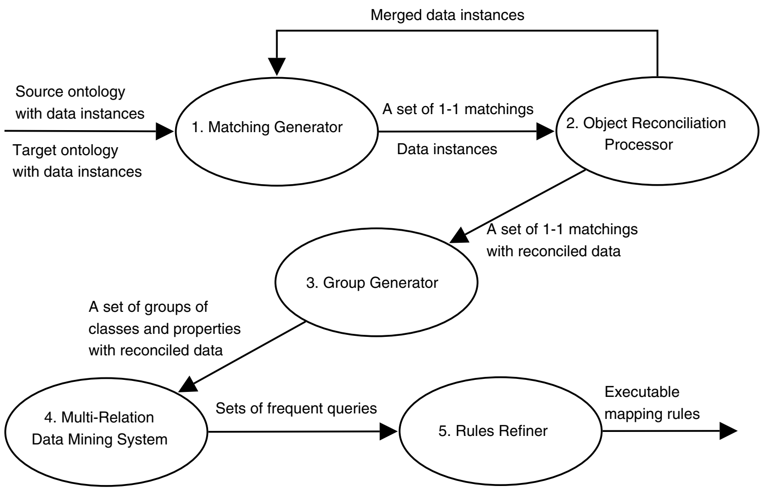

Merged data instances
Source ontology with data instances
Target ontology with data instances
1. Matching Generator
A set of 1-1 matchings
Data instances
2. Object Reconciliation Processor
A set of 1-1 matchings with reconciled data
3. Group Generator
A set of groups of classes and properties with reconciled data
4. Multi-Relation Data Mining System
Sets of frequent queries
5. Rules Refiner
Executable mapping rules

Fig. 1. System Architecture

1. Matching Generator (MG): It takes the source and target ontologies with their data instances as input. Different matching system could be plugged in this component. The output of MG is 1-1 matching pairs between classes and properties. If no new matchings can be found, MG will pass this information to the next component, Object Reconciliation Processor. It also passes the data instances.
2. Object Reconciliation Processor (ORP): This component is designed to reconcile instances which refer to the same real world entities. If there are new matchings from Matching Generator, ORP will try to reconcile more instances. Otherwise it passes the 1-1 matchings and the reconciled data to the next component, Group Generator.
3. Group Generator (GG): The matchings are not always 1-1 matchings. GG combines close related 1-1 property matching pairs, class matching pairs and their instances together as a group. For every group, GG generates a set of input data for multi relation data mining system.
4. Multi-Relation Data Mining System (MRDMS): We borrow some ideas of FARMER system to mine *frequent queries*. This system requires certain input format and it can find interesting frequent queries. Since every group generated by GG is comparably small, the search space is not a concern.

5. Rules Refiner (RR): Generating executable mapping rules from frequent queries is a natural extension. However, it is not suitable to set a fixed threshold for support and confidence. The threshold can be different for different cases. RR will filter out the rules which are distinctly incorrect and keep the rest.

3.2 Approach Overview with a Simple Example

We give the overview of our approach with a simple mapping example based on the People Ontology[2] from UMD and the Person & Employee Ontology[3] from CMU. And we also use these two ontologies to illustrate our system in the following sections.

The first step of our approach is to find corresponding classes and properties. Currently we use a string matching algorithm [30] to get 1-1 matchings. For example, two properties $name(Person, String)$ (from the People Ontology) and $name\_person(Person, String)$ (from the Person& Employee Ontology) have similar names and are considered matched. Before finding (mining) the semantic mapping of these two concepts, there is one problem that must be solved. Instances from different ontologies may represent the same object but have different names. For example, both of the instance "person001" described by the People Ontology and the instance "p001" described by the Person& Employee Ontology may actually refer the same person (e.g., `Han Qin`). Therefore we have to reconcile them by renaming "p001" to "person001" or vice-versa. This is actually an object reconciliation problem. Giving the matchings we can use some mature object reconciliation algorithm to reconcile the instances. The reconciliation result can help find new matchings. Therefore this is an iterative process between ontology matching and object reconciliation.

In the next step, we generate matching groups of classes and properties. Then we generate the input for the data multi-relational data mining (MRDM) system. Finally MRDM system will mine the *frequent queries* based on the input data. A query is a logical expression of the form $? - P_1, ..., P_n$, which contains an atom key used for counting. For one query if the number of answers of atom key exceeds the threshold, we call this query as frequent query. For this example, one frequent query could be:

?- @UMD:Person(x), @UMD:name(x,y), @CMU:name_person(x,y)

where $Person$ is the atom key and we use "*@UMD:*" and "*@CMU:*" to represent prefixes of the People and Person& Employee ontologies respectively. Since we are interested in the frequent queries related to *name* and *name_person*, we will derive one rule from this frequent query:

$$\forall x, y \ @UMD{:}Person(x) \wedge @UMD{:}name(x, y) \rightarrow @CMU{:}Person(x) \wedge @CMU{:}name\_person(x, y)$$

[2] http://www.cs.umd.edu/projects/plus/DAML/onts/personal1.0.daml

[3] http://www.daml.ri.cmu.edu/ont/homework/atlas-cmu.daml

4 Matching and Object Reconciliation

In this section, we introduce an iterative process between the Matching Generator and Object Reconciliation Processor. We still use some examples based on the People and Person&Employee ontologies to help demonstrate how this process works.

4.1 Basic Name Matching

We begin from finding class matching pairs and property matching pairs based on their names. We make use of "Iterative SubString Matching Algorithm" [30] to calculate the similarity of names of each pair. We basically examine every pair of classes and properties from both ontologies. For example, if N is the number of classes in the source ontology and M is the number of classes in the target ontology, there are $N * M$ potential class matching pairs. For each pair we can get a similarity score and we also set a threshold for name similarity to get class matching pairs. Similarly we find some property matching pairs. Other existing matching approaches (e.g., synonym-based approaches by using Wordnet[15]) can also be used in this step to help find more matchings.

4.2 Datatype Property Matching

There are some property matching pairs which can strengthen our confidence about potential class matching pairs. For example, one kind of OWL properties is datatype properties and the range is a data type, such as string and number. Datatype property matching pairs can support our system to match a class pair with higher confidence. We can use the types of property arguments to find them. Given the datatype property pair $p(X, Y)$ and $q(U, V)$, where X and U are classes and Y and V are data types, if there is a class matching pair $X \leftrightsquigarrow U$ and Y is the same data type as V, we consider $p(X, Y) \leftrightsquigarrow q(U, V)$ as one potential datatype property matching related to the class matching pair $X \leftrightsquigarrow U$. Not only the name similarity of p and q is needed to make more confidence of $X \leftrightsquigarrow U$, but also the data value similarity of p and q. One potential datatype property matching will be verified by data value similarity which will be further discussed in Section 4.3. An example is that $\mathit{@UMD{:}name}(Person, String) \leftrightsquigarrow \mathit{@CMU{:}name\_person}(Person, String)$ is one potential datatype property matching based on class matching pair $\mathit{@UMD{:}Person} \leftrightsquigarrow \mathit{@CMU{:}Person}$. The higher the similarity of datatype property matching pairs are, the more confidence this class matching pair has. Support of class matching pair is calculated according to the following equation:

$$Support(X \leftrightsquigarrow U) = \Sigma DatatypePropertyPairSimilarity \qquad (1)$$

Datatype property pair similarity is the sum of name similarity and data similarity, which will be further discussed in Section 4.3.

Another problem we should cope with is that two classes may have totally unrelated names, but they represent the same concept. One clue to handle this case is actually from datatype property matchings. If several property matching pairs indicate that class X and class U should be a pair, we can assume $X \leftrightsquigarrow U$ is one class matching pair and add those property matching pairs as its datatype property matchings.

4.3 Data Similarity of Datatype Property Pairs

Having a similar name does not necessarily mean that two properties definitely represent the similar concept. Therefore, calculating data similarity of property matching pairs is necessary. Note that data similarity is based on the assumption that the data instances of two ontologies overlap at a relatively high level, otherwise we can not benefit from data level examination. The data similarity can help verify both property matching pairs and those class matching pairs that they are related to. For an existing property matching pair $p(X, Y) \leftrightsquigarrow q(U, V)$, we can calculate the data similarity of p and q by using the following formula:

$$DataSimilarity(p(X, Y) \leftrightsquigarrow q(U, V)) = \frac{2 * |same\_pairs(Y, V)|}{|p(X, Y)| + |q(U, V)|} \qquad (2)$$

where $|same\_pairs(Y, V)|$ is the number of the instance pairs which have the same integer value or very similar string value, and $|p(X, Y)|$ and $|q(U, V)|$ stand for the number of instances of property $p(X, Y)$ and $q(U, V)$.

Then we can calculate the total similarity of each class matching pair. For one class matching pair $X \leftrightsquigarrow U$, the similarity is the sum of name similarity and its support. If both clues show that this matching pair is not correct, we remove it from the class matching pair list.

4.4 Object Reconciliation

After we get the selected class matching pairs, we adopt an object reconciliation algorithm developed by Dong, Halevy and Madhavan in [10] to reconcile the instances of classes from two ontologies. The original algorithm considers the relationship of matching pairs and determines whether two data objects in different databases represent the same real-world entity. We successfully use the idea in our ontology mapping examples.

We do not want to repeat the detail algorithm since it can be found in the paper [10]. In a summary, for all the possible instance pairs, we can draw a similarity graph and calculate the similarity between them. The formula we use is a simple aggregation of the similarity of datatype matching pairs. For example, Figure 2 shows one positive example and one negative example for People and Person&Employee ontology mapping. Node (person001, p001) has a high similarity and is considered as reconciled. Node (person002, p003) has a comparably low similarity and is considered as not reconciled.

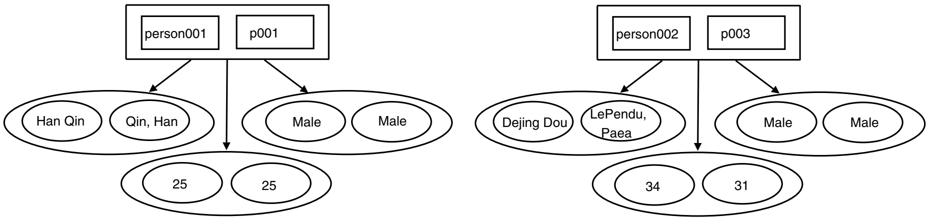


Fig. 2. Positive and negative object reconciliation examples

4.5 Object Property Matching

In OWL, object properties represent the relationships between two classes, such as the property $alumnus(Organization, Person)$ from the UMD People ontology. Similar to datatype property matching pairs, the object property matching pairs may have similar property name or not. However, it is harder to find object property matchings because their data instances can not help before object reconciliation process is performed. Therefore, only after we reconcile some data instances we can calculate the data similarity of object property matching pairs. Note that this kind of matching pairs have two ways to match: given $p(X, Y) \leftrightsquigarrow q(U, V)$, the first matching is $X \leftrightsquigarrow U, Y \leftrightsquigarrow V$ while the other is $X \leftrightsquigarrow V, Y \leftrightsquigarrow U$. When we calculate the data similarity of object property matchings, whether it is cross matched should be labeled. Similar to data property matching pairs, we give the following formula:

$$DataSimilarity(p(X,Y) \leftrightsquigarrow q(U,V)) = \frac{|same\_pairs(X,U)| + |same\_pairs(Y,V)|}{|p(X,Y)| + |q(U,V)|} \tag{3}$$

After Object Reconciliation Processor executes in each iteration, the Matching Generator tries to create new object property matching pairs based on the object reconciliation results. Figure 3 shows an example of this iterative process: after the first time Object Reconciliation Processor executed based on *@UMD:Person* $\leftrightsquigarrow$ *@CMU:Person*, the system found that "person001" is the same entity as "p001". Given this result, Matching Generator will find a new object property matching pair *@UMD:alumnus* $(Organization, Person)$ $\leftrightsquigarrow$ *@CMU:has_employes*$(Organization, Person)$, since there exists two class matching pairs *@UMD:Organization* $\leftrightsquigarrow$ *@CMU:Organization* and *@UMD:Pe− rson* $\leftrightsquigarrow$ *@CMU:Person* and the data similarity of this property matching pair also shows these two properties should be matched. This new property matching pair will be returned to the system to suggest reconcile more data instances related to *@UMD:Organization* $\leftrightsquigarrow$ *@CMU:Organization*, such as "uo_cs" and "CS_dept". This reconciliation result may help Matching Generator to find more object property matching pairs related to *@UMD:Organization* and *@CMU:Organization*. The process will be end if no new object property matchings can be found. At the end we have the complete graph of class matching pairs and property matching pairs as shown in Figure 4.

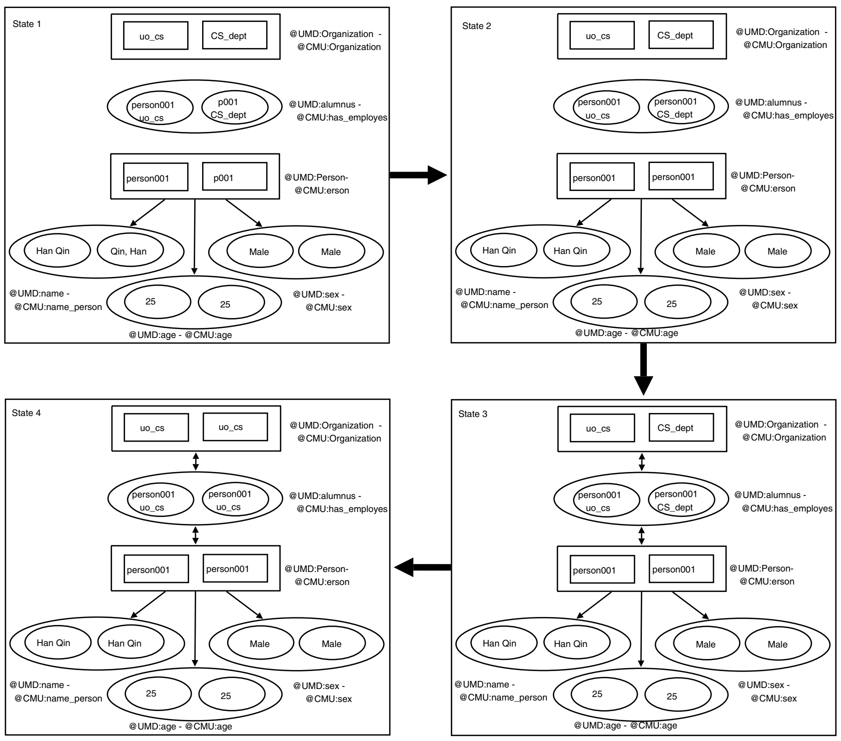


Fig. 3. The Iterative Process Of the People and Person&Employee Ontology Mapping Example: After Matching Generator creates new matchings in each iteration, Object Reconciliation Processor can reconcile more data instances and then help MG to find more new matchings. This process will terminate when MG can not find any new matchings.

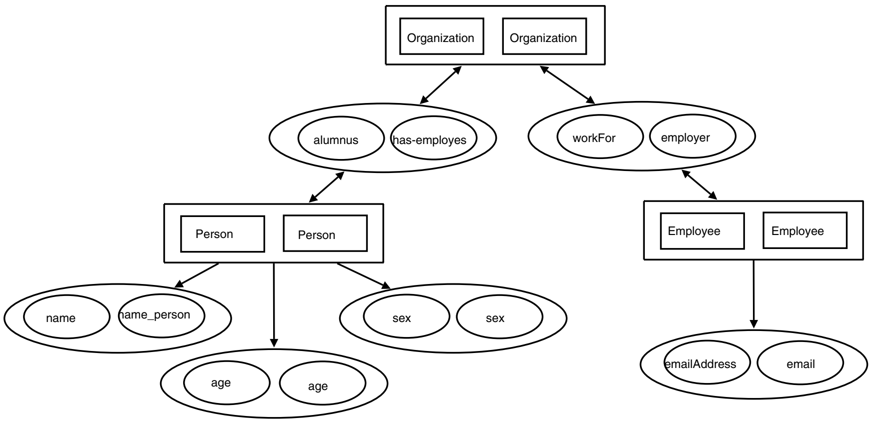


Fig. 4. The Graph Of Class Matching Pairs and Property Matching Pairs

5 Semantic Mapping Rule Mining

Based on the matching and object reconciliation results, we first generate group matchings and then use data mining techniques to discover final mapping rules.

5.1 Matching Groups

In this step, we tackle the problem of reducing the search space of the MRDM systems by generating groups based on 1-1 matchings. The simplest case is to put some of them together as a group based on related classes or properties, such as that *@UMD:name(Person, String)* $\leftrightsquigarrow$ *@CMU:name_person(Person, String)*, *@UMD:Person* $\leftrightsquigarrow$ *@CMU:Person* compose one group. The basic grouping rule we have used is:

Basic Grouping Rule: *For one property matching pair @source:p(X,Y)* $\leftrightsquigarrow$ *@target:q(U,V), if there are class matching pair @source:X* $\leftrightsquigarrow$ *@target:U and @source:Y* $\leftrightsquigarrow$ *@target:V, we generate one group which includes all these three matchings.*

Note that if *@source:Y* $\leftrightsquigarrow$ *@target:V* is a pair of string or number, we do not have to really put them in. This rule can cover most of the 1-1 property matching pairs, especially datatype property matching pairs.

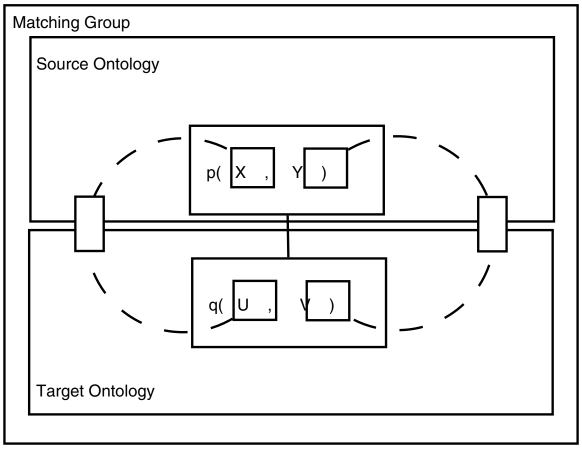


Fig. 5. Matcha ming Group Based On One Property Matching Pair

A complex case is that when we check *@source:p(X,Y)* $\leftrightsquigarrow$ *@target:q(U,V)* we only know that one class matching pair (i.e., *@source:X* $\leftrightsquigarrow$ *@target:U* or *@source:Y* $\leftrightsquigarrow$ *@target:V*) exists. Suppose that *@source:X* $\leftrightsquigarrow$ *@target:U* exists, then obviously this group is not complete yet. Therefore, we have to include other properties in this group to make it complete. The basic idea is: assume we can find *@source:Z* $\leftrightsquigarrow$ *@target:V* and in the source ontology there is one property *@source:r(Y,Z)*, then we can make the guess and include *@source:r(Y,Z)* in this group. Another possible solution is try to find *@target:t(W, V)* if *@source:Y* $\leftrightsquigarrow$ *@target:W* exists. To sum up, we have to find some connection between *@source:Y* $\leftrightsquigarrow$ *@target:V*. One example in Figure 6 is *@UMD:workphone (Person, String)* $\leftrightsquigarrow$ *@CMU:phone(Office, String)*. In the source ontology (i.e., CMU), we can find one related property *@CMU:office(Employee, Office)*, which connects *@UMD:Person* with *@CMU:Office* since *@CMU:Employee* is a subclass of *@CMU:Person* and *@UMD:Person* $\leftrightsquigarrow$ *@CMU:Person* is a class matching pair. Thus this group contains three properties.

For more general cases, we should find connections between both *@source:X* $\leftrightsquigarrow$ *@target:U* and *@source:Y* $\leftrightsquigarrow$ *@target:V*. Figure 5 shows what a complete group is. The dashed line refers to zero or several predicates, super-sub class relationships or class matching pairs. Based on this, we can draw the general group rule:

General Grouping Rule: *For one property matching pair, such as @source: p(X, Y)* $\leftrightsquigarrow$ *@target:q(U, V), if we do not have class matching pair @source:X* $\leftrightsquigarrow$ *@target:U (or @source:Y* $\leftrightsquigarrow$ *@target:V), we can search among the properties and class matching pairs to find a connection path from @source:X to @target:U (or from @source:Y to @target:V). In the path there must exist one class matching pair that connects the source and target ontologies.*

Discovering the path is the key step for finding group matchings. And the class matching pair that connects the source and target ontologies is the key class matching pair. To find the path from *@source:X* to *@target:U*, if we can find the key class matching pair *@source:A* $\leftrightsquigarrow$ *@target:B*, the path contains the path from *@source:X* to *@source:A*, *@source:A* $\leftrightsquigarrow$ *@target:B* and the path from *@target:B* to *@target:U*. And we propose Algorithm 1 to find the key class matching pair:

Algorithm 1. Searching Key Class Matching Pair

```
Input: class X from source ontology, class U from target ontology, properties of both
    ontologies except p(X, Y) and q(U, V), class matching pair set, super-sub class
    relationship in two ontologies.
Output: the key class matching pair
  Initialize source class set with X
  Initialize target class set with U
  while There does not exist class matching pair A ⇜⇝ B, where A belongs to source
  class set and B belongs to target class set. do
      For all t(H, K), t(K, H), superclass(H, K) and subclass (H, K) where H is in source
      class set, add K into source class set.
      For all r(N, M), r(M, N), superclass(M, N) and subclass (M, N) where M is in
      target class set, add N into target class set.
      If no new classes is added, return No_pair.
  end while
  Return A ⇜⇝ B
```

5.2 Generating Mapping Rules

The first step of this part is to discover frequent queries. The algorithm should take a set of predicates and data instances as input, build the search space and finally output a set of queries with high support. We consider classes and properties of ontology as unary or binary predicates. FARMER [25] system can be used for this goal. However, FARMER requires users to specify the input/output type of each argument of the predicates. To make the whole process as automatically as possible we borrow some ideas of FARMER but create a new algorithm (see Algorithm 2) instead in our implementation.

Algorithm 2. Generating frequent queries

```
Input: A matching group G. Data instances of all the predicates in G.
Output: Frequent queries with their support.
  Create the first query with the key class matching pair of G.
  while Not all predicates of the source ontology is added to the first query. do
    Suppose the type of last argument of the first query is T. Then find predicate P
    (in the source ontology) which has first argument type T. Add P to the end of the
    first query.
  end while
  Calculate the support of the first query.
  Create the second query as a copy of the first query.
  while Not all predicates of the target ontology is added to the second query. do
    Suppose the type of last argument of the second query is V. Then find predicate
    Q (in the target ontology) which has first argument type V. Add Q to the end of
    the second query.
  end while
  Calculate the support of the second query.
  Return two queries with their support.
```

The next step of rule generation is a natural extension from frequent queries. The Rule Refiner can generate mapping rules based on frequent queries and class matching pairs. For example, the output of matching group `G` is:

```
? - Person(V0N0),name(V0N0,V1N0) support: 100
? - Person(V0N0),name(V0N0,V1N0),name_person(V0N0,V1N0) support: 95
```

With class matching pair *@UMD:Person* $\leftrightsquigarrow$ *@CMU:Person*, a rule like $\forall x, y$ *@UMD:Person*$(x) \wedge$ *@UMD:name*$(x, y) \rightarrow$ *@CMU:Person*$(x) \wedge$ *@CMU:name_person*(x, y) can be generated.

This process is similar to a typical multi-relational data mining process. The main difference is that the information of each matching group helps to reduce the search space of query searching. We consider rules with extremely low support and confidence as distinctly incorrect. Thus we set a very low threshold and keep the rest rules.

6 Case Study

6.1 People vs. Person and Employee Ontology

The first case we test is the complete UMD People Ontology and CMU Person & Employee Ontology mapping example. The partial examples we give in previous sections are from this case. Figure 6 shows part of two ontologies and some human labeled matchings. Dotted lines refer to property matchings and dashed lines refer to class matchings.

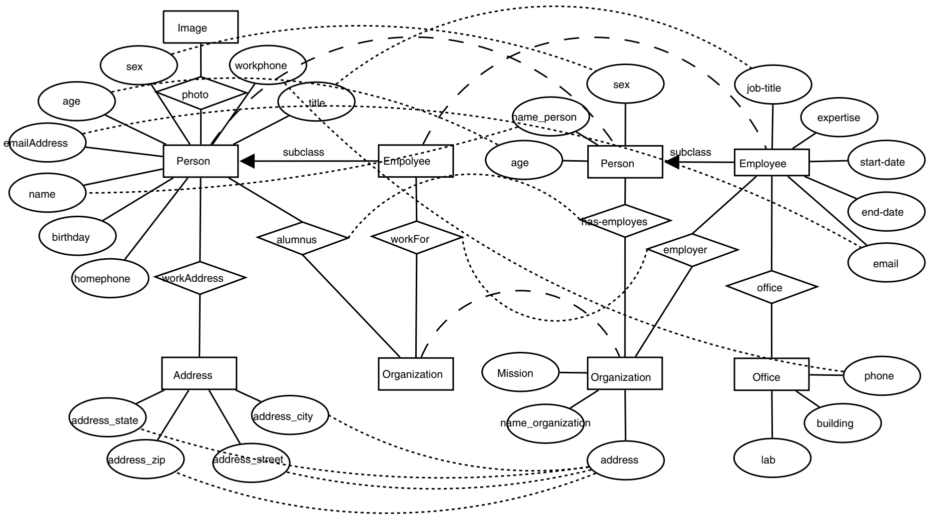


Fig. 6. UMD and CMU person ontology

The Matching Generator and Object Reconciliation Processor gives three class matching pairs and 21 property matching pairs. For some matching pairs, we cannot find any complete group. Group Generator actually outputs 17 matching groups. Finally, our system generates 8 rules. With human observation, we can get 9 rules manually. Therefore, the performance of our system in this case is satisfying as 8 out of 9. We can represent the mapping rules in first order logic syntax, such as: $\forall x, y\ Employee(x) \wedge worksfor(x, y) \rightarrow Employee(x) \wedge employer(x, y)$. Also we can represent these rules in our Web-PDDL or different potential standard mapping languages. For example, we can represent rules with Datalog:

```
emailAddress(A, B) :- Person(A), email(A,B).
```

Or we can represent rules with SWRL as following:

```
<ruleml:imp>
 <ruleml:_rlab ruleml:href="#Rule2"/>
```

```
  <ruleml:_body>
    <swrlx:classAtom>
      <owlx:Class owlx:name="&UMD;Person" />
      <ruleml:var>x</ruleml:var>
    </swrlx:classAtom>
    <swrlx:individualPropertyAtom  swrlx:property="&UMD;name">
      <ruleml:var>x</ruleml:var>
      <ruleml:var>y</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom  swrlx:property="&CMU;name_person">
      <ruleml:var>x</ruleml:var>
      <ruleml:var>y</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

The reason we missed one rule is: our system can find the matchings which contain *@UMD:address_state*, *@UMD:address_city*, *@UMD:address_zip*, *@UMD:address_steet* and *@CMU:address*. However, data mining system cannot introduce functions, therefore we do not derive any rule for these matchings successfully. This missing rule needs a concatenation function to combine *@UMD:address_state*, *@UMD:address_city*, *@UMD:address_zip*, *@UMD: address_steet* to *@CMU:address*.

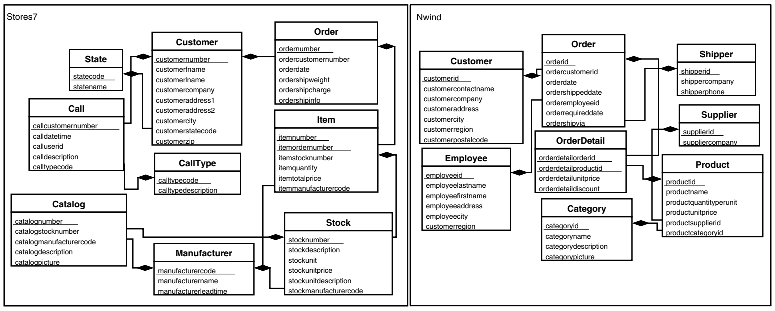


Fig. 7. Stores7 and Northwind schema

6.2 Online Sale Databases

Our approach can also be applied to databases. Our previous research [11,12] demonstrates a way to convert database schemas to ontologies. In this case, we consider two database schemas: Stores7 from IBM Informix4[4] and Northwind from Microsoft. Both of them are from online sales domain and have related concepts, such as Customer, Order and etc. Figure 7 shows the schemas of two

[4] http://www.ibm.com/software/data/informix/

databases. We have used these two databases to test our OntoGrate system but we used human-specified mappings in [11,12].

Humans can find 18 rules for these two database schemas. And our system discovers 4 class matching pairs and 201 property matching pairs. Then finally 16 rules are obtained by our system. Two rules of our output are incorrect. The reason of incorrectness is also that the MRDM system does not introduce functions, same as the address matching group problem addressed in section 6.1.

7 Conclusion and Future Work

We present a highly automatic approach which combines ontology matching, object reconciliation and multi-relational data mining to discover the executable mapping rules between given source and target ontologies from same domain. Our main novel contributions are:

1. We propose an iterative process: basic matchings can be used to guide object reconciliation and the result of object reconciliation can guide to find new matchings. This process also help verify existing matchings.

2. We propose a way to combine matching pairs to form matching groups, which is used to generate queries and mapping rules.

3. We use a data mining approach to find frequent queries and then convert them to mapping rules. We use group matchings to reduce the search space.

Our approach relies on both data instances and ontologies, and thus a constraint is that we must need ontologies with related data instances. Our system cannot guarantee 100% accurately generated rules. If users need 100% perfect results, human effort is surely needed. There are still a lot of interesting problems we cannot solve yet. The first problem we are going to solve is to discover mappings with new functions, such as $\forall x, y, z Person(x) \wedge city\_address(x, y) \wedge street\_address(x, z) \rightarrow address(x, concatenate(y, z))$. Then we will consider how to automatically evaluate an ontology mapping system without human-specified results and how to manage the mapping rules in the scenario that ontologies or database schemas keep changing.

References

1. Semantic Web Rule Language, http://www.w3.org/Submission/SWRL/
2. An, Y., Borgida, A., Mylopoulos, J.: Constructing complex semantic mappings between XML data and ontologies. In: International Semantic Web Conference, pp. 6–20 (2005)
3. An, Y., Borgida, A., Mylopoulos, J.: Inferring complex semantic mappings between relational tables and ontologies from simple correspondences. In: OTM Conferences (2), ODBASE, pp. 1152–1169 (2005)
4. Bayardo, R.J., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., Rashid, M., Rusinkiewicz, M., Shea, R., Unnikrishnan, C., Unruh, A., Woelk, D.: InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In: SIGMOD 1997. Proceedings of the 1997 ACM SIGMOD international conference on Management of data, pp. 195–206. ACM Press, New York (1997)

5. Bruijn, J.D., Polleres, A.: Towards an Ontology Mapping Specification Language for the Semantic Web. Technical report, Digital Enterprise Research Institute (June 2004)
6. Dehaspe, L., Toivonen, H.: Discovery of frequent datalog patterns. Data Min. Knowl. Discov. 3(1), 7–36 (1999)
7. Dhamankar, R., Lee, Y., Doan, A., Halevy, A.Y., Domingos, P.: iMAP: Discovering Complex Mappings between Database Schemas. In: Proceedings of the ACM Conference on Management of Data, pp. 383–394. ACM Press, New York (2004)
8. Doan, A., Domingos, P., Halevy, A.Y.: Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In: Proceedings of the ACM Conference on Management of Data, ACM Press, New York (2001)
9. Doan, A., Madhavan, J., Domingos, P., Halevy, A.Y.: Learning to Map Between Ontologies on the Semantic Web. In: WWW. International World Wide Web Conferences, pp. 662–673 (2002)
10. Dong, X., Halevy, A., Madhavan, J.: Reference reconciliation in complex information spaces. In: SIGMOD 2005. Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 85–96. ACM Press, New York (2005)
11. Dou, D., LePendu, P.: Ontology-based Integration for Relational Databases. In: SAC 2006. Proceedings of the 2006 ACM symposium on Applied computing, pp. 461–466. ACM Press, New York (2006)
12. Dou, D., LePendu, P., Kim, S., Qi, P.: Integrating Databases into the Semantic Web through an Ontology-based Framework. In: SWDB 2006. Proceedings of the third International Workshop on Semantic Web and Databases, p. 54 (2006)
13. Dou, D., McDermott, D.V., Qi, P.: Ontology Translation on the Semantic Web. Journal of Data Semantics 2, 35–57 (2005)
14. Dragut, E., Lawrence, R.: Composing mappings between schemas using a reference ontology. In: ODBASE. Proceedings of International Conference on Ontologies, Databases and Application of Semantics (2004)
15. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
16. T. Gene Ontology Consortium. Creating the Gene Ontology Resource: Design and Implementation. Genome Research, 11(8), 1425–1433 (2001)
17. Haas, L.M., Hernandez, M.A., Ho, H., Popa, L., Roth, M.: Clio Grows Up: From Research Prototype to Industrial Tool. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 805–810. ACM Press, New York (2005)
18. Hu, W., Qu, Y.: Block matching for ontologies. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 5–9. Springer, Heidelberg (2006)
19. Lindberg, D., Humphries, B., McCray, A.: The Unified Medical Language System. Methods of Information in Medicine 32(4), 281–291 (1993)
20. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic Schema Matching with Cupid. In: Very Large Data Bases (VLDB) Conference, pp. 49–58 (2001)
21. Maedche, A., Motik, B., Silva, N., Volz, R.: MAFRA - A MApping FRAmework for Distributed Ontologies, pp. 235–250 (2002)
22. McGuinness, D.L., Fikes, R., Rice, J., Wilder, S.: The Chimaera Ontology Environment. In: Proceedings of the National Conference on Artificial Intelligence, pp. 1123–1124 (2000)

23. Mena, E., Kashyap, V., Sheth, A.P., Illarramendi, A.: Observer: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In: CoopIS, pp. 14–25 (1996)
24. Miller, R.J., Hernández, M.A., Haas, L.M., Yan, L.-L., Ho, C.T.H., Fagin, R., Popa, L.: The clio project: Managing heterogeneity. SIGMOD Record 30(1), 78–83 (2001)
25. Nijssen, S., Kok, J.N.: Efficient frequent query discovery in farmer. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) PKDD 2003. LNCS (LNAI), vol. 2838, pp. 350–362. Springer, Heidelberg (2003)
26. Noy, N.F.: Semantic Integration: A Survey Of Ontology-Based Approaches. SIGMOD Record 33(4), 65–70 (2004)
27. Noy, N.F., Musen, M.A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: Proceedings of the National Conference on Artificial Intelligence, pp. 450–455 (2000)
28. Quinlan, J.R., Cameron-Jones, R.M.: Foil: A midterm report. In: Brazdil, P.B. (ed.) ECML 1993. LNCS, vol. 667, pp. 3–20. Springer, Heidelberg (1993)
29. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB J. 10(4), 334–350 (2001)
30. Stoilos, G., Stamou, G.B., Kollias, S.D.: A string metric for ontology alignment. In: International Semantic Web Conference, pp. 624–637 (2005)
31. Stuckenschmidt, H., Uschold, M.: Representation of semantic mappings. Semantic Interoperability and Integration (2005)
32. Wache, H., Vogele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hubner, S.: Ontology-based integration of information: A survey of existing approaches. In: IJCAI 2001. Workshop: Ontologies and Information Sharing, pp. 108–117 (2001)

Interoperability of XML Schema Applications with OWL Domain Knowledge and Semantic Web Tools

Chrisa Tsinaraki and Stavros Christodoulakis

TUC/MUSIC, Technical University of Crete Campus, 73100 Kounoupidiana, Crete, Greece
{chrisa, stavros}@ced.tuc.gr

Abstract. Several standards are expressed using XML Schema syntax, since the XML is the default standard for data exchange in the Internet. However, several applications need semantic support offered by domain ontologies and semantic Web tools like logic-based reasoners. Thus, there is a strong need for interoperability between XML Schema and OWL. This can be achieved if the XML schema constructs are expressed in OWL, where the enrichment with OWL domain ontologies and further semantic processing are possible. After semantic processing, the derived OWL constructs should be converted back to instances of the original schema. We present in this paper XS2OWL, a model and a system that allow the transformation of XML Schemas to OWL-DL constructs. These constructs can be used to drive the automatic creation of OWL domain ontologies and individuals. The XS2OWL transformation model allows the correct conversion of the derived knowledge from OWL-DL back to XML constructs valid according to the original XML Schemas, in order to be used transparently by the applications that follow XML Schema syntax of the standards.

Keywords: Interoperability, Standards, XML Schema, OWL, Ontologies.

1 Introduction

The development of the Web and the emergence of advanced network infrastructures that allow the fast and efficient information delivery allow the end-users to access Web documents and Web applications. In such an open environment, the applications developed by different vendors interoperate on the basis of the emergent standards. The default standard for data exchange in the Internet today is the *eXtensible Markup Language (XML)* [3], which allows the representation of structured Web documents. The classes of the Web documents are described using the *XML Schema Language* [6], which uses XML syntax and supports very rich structures and datatypes for XML documents. Due to its structural capabilities and the central role in the data exchange in the Internet, many standards in different application areas are directly specified in XML Schema. Examples include several important standards in multimedia, like MPEG-7 [5] and MPEG-21 [15], in e-learning, like IEEE LOM [11] and SCORM [1], in Digital Libraries, like METS [10], and many others.

However, several Web applications that utilize XML-based standards would benefit from advanced semantic support. These applications typically need to integrate domain ontologies and do further semantic processing including reasoning within the constructs that the standards provide. For example, consider the MPEG-7 applications. MPEG-7, which is described using XML Schema syntax, provides rich language constructs for describing the structure and the content of multimedia data such as video. Retrieval, browsing, personalization and delivery applications in MPEG-7 would benefit from the use of domain knowledge, and MPEG-7 includes general-purpose constructs that could be used for describing domain knowledge, albeit in a rather cumbersome way [18]. Programmers that are going to introduce domain knowledge in MPEG-7 are much more likely to be familiar with the ontology description mechanisms of the *Web Ontology Language (OWL)* [14] than those of MPEG-7. In addition, some MPEG-7 applications, like, for example, knowledge acquisition from video streams, may significantly benefit from the use of logic-based reasoners such as the ones available for OWL. There is then a strong motivation for some MPEG-7 applications to be able to work with the semantics of MPEG-7 expressed in OWL and integrated with OWL domain ontologies. This way, additional acquired knowledge expressed in OWL such as metadata acquired during knowledge acquisition from video streams can be encoded using the semantics of the standard. The resulting knowledge should be converted back to standard MPEG-7/XML constructs in order to be used transparently by applications that follow the standard.

We present in this paper XS2OWL, a model and system software that allow applications using XML Schema based standards to use the Semantic Web methodologies and tools while maintaining the compatibility with the XML schema versions of the standards. With XS2OWL we express each XML Schema based standard in OWL-DL as a *Main Ontology* so that the constructs of the standard become Semantic Web objects. The main ontology allows integrating the constructs of the standard with domain knowledge expressed in the form of OWL domain ontologies. It also allows all the OWL-based Semantic Web tools, including reasoners, to be used with the standard-based descriptions. The integration of the domain ontologies and the utilization of the OWL-based Semantic Web tools may result in deriving additional knowledge useful to the applications, expressed as OWL individuals. The XS2OWL transformation model also supports the conversion of the OWL constructs back to the XML Schema based constructs of the standard for use by all the applications that are compatible with the original XML Schemas. This is achieved through a *Mapping Ontology* that systematically captures the semantics of the XML Schema constructs that cannot be directly captured in the main ontology.

Very limited research has been reported in the literature in this area. We had first observed the need for such methodology and software in conjunction with MPEG-7 [16, 17, 18]. For the use of some MPEG-7 applications we developed a manual methodology for expressing the MPEG-7 semantics in OWL. An Upper OWL-DL ontology capturing the *MPEG-7 Multimedia Description Schemes (MDS)* [13] and the *MPEG-21 Digital Item Adaptation (DIA) Architecture* [12] was defined [16]. This ontology could then be extended with domain ontologies. A soccer ontology and a Formula 1 ontology have been developed as extensions of the Upper ontology [17] for capturing knowledge from video streams in these two domains. We also defined and implemented a set of transformation rules [17] that allow the transformation of the OWL metadata (individuals) that describe the multimedia content defined using the Upper ontology and the domain ontologies into the original MPEG-7/21 constructs.

The transformation rules relied on a mapping ontology that systematically captured the information of the MPEG-7/21 schemas that cannot be captured in the Upper ontology. Although this work is an important motivating example for the need of the general-purpose mechanism described in this paper, it is only a special case applicable to MPEG-7/21 and to the applications that use them. In addition, the conversion of the XML Schema constructs describing the MPEG-7/21 to OWL-DL was done manually and the transformation back to XML Schema based syntax was focusing on the needs of MPEG-7/21.

A methodology and a tool have been developed in the context of the MapOnto project [2] for the heuristic definition of complex semantic mappings between the attributes of an XML Schema and the datatype properties of an OWL ontology. The tool input consists of the XML Schema, the OWL ontology and a set of correspondences between the attributes of the former and the datatype properties of the later.

In a recent work [7], almost automatic one-way transformation of XML Schema constructs to OWL constructs has been proposed. The transformations produce OWL-Full ontologies that partially capture the XML Schema semantics. The methodology proposed in [7] looses information during the transformation process from XML Schema to OWL, and it can not be used to transform OWL individuals which may be produced later back to the original XML Schema constructs. This is however needed in all the applications mentioned above. In addition, human intervention is needed in order to preserve the validity of the ontologies produced when homonym top-level XML Schema constructs exist in the source XML Schema (i.e. elements, attributes, types or groups with the same name). Finally, some transformations in [7] do not follow closely the semantics of the XML Schema.

In contrast to the work reported in [7], the XS2OWL model presented in this paper encapsulates methodology and rules that allow automatically transforming the XML Schema constructs to OWL-DL constructs (not OWL-Full). Thus, it guarantees computational completeness and decidability of reasoning in the OWL ontologies produced. In addition, the XS2OWL model allows the representation of all the knowledge needed to transform the individuals generated or added later on to the OWL ontologies back to the original XML Schema based constructs. This way, they can be used by the standard-compatible applications. Finally, we have implemented and extensively tested the XS2OWL model with very large standards based on XML, and verified manually and automatically the correctness of the model and software.

The rest of the paper is structured as follows: In section 2 we provide background information. The XS2OWL transformation model and system are outlined in section 3 and the XS2OWL model is detailed in section 4. In section 5 we present the XS2OWL model evaluation. The paper conclusions and our future research directions are presented in section 6.

2 Background

In this section we present some background information needed in the rest of the paper, including the *XML Schema Language* and the *Web Ontology Language (OWL).*

The XML Schema Language. The *XML Schema Language* [6] allows the definition of classes of XML documents using XML syntax and provides datatypes and rich structuring capabilities. An XML document is composed of *elements*, with the root element delimiting the beginning and the end of the document. The XML Schema elements belong to XML Schema *types*, specified in their "type" attribute, and are distinguished into complex and simple elements, depending on the kind (simple or complex) of the types they belong to. Reuse of element definitions is supported by the *substitutionGroup* attribute, which states that the current element is a specialization of another element. The elements may either have a predefined order (forming XML Schema *sequences*) or be unordered (forming XML Schema *choices*). The main difference between sequences and choices is that all the sequence items must appear within the containing sequence in their specified order, while the choice items may appear at any order. Both sequences and choices may be nested. The minimum and maximum number of occurrences of the elements, choices and sequences are specified, respectively, in the "minOccurs" and "maxOccurs" attributes (absent "minOccurs" and/or "maxOccurs" correspond to values of 1). Reusable complex structures, combining sequences and choices, may be defined as *model groups*.

The *simple XML Schema types* are usually defined as restrictions of the basic datatypes provided by XML Schema (i.e. strings, integers, floats, tokens etc.). Simple types can neither contain elements nor carry attributes. The *complex XML Schema types* represent classes of XML constructs that have common features, represented by their elements and *attributes*. The attributes describe features with values of simple type and may form *attribute groups* comprised of attributes that should be used simultaneously. The elements represent features of the complex XML Schema types with values of any type. Default and fixed values may be specified for both attributes and simple type elements, in the *default* and *fixed* attributes respectively. Inheritance is supported for both simple and complex types, and the base types are referenced in the "base" attribute of the type definitions.

All the XML Schema constructs may have textual annotations, specified in their "annotation" element. The top-level XML Schema constructs (attributes, elements, simple and complex types, attribute and model groups) have unique *names* (specified in their "name" attribute). The nested elements and attributes have unique names in the context of the complex types in which they are defined, while the nested (complex and simple) types are unnamed. All the XML Schema constructs may have unique identifiers (specified in their "id" attribute). The top-level constructs may be referenced by other constructs using the "ref" attribute.

The Web Ontology Language (OWL). The *Web Ontology Language (OWL)* [14] is the dominant standard in ontology definition. OWL has been developed according to the description logics paradigm and uses *RDF (Resource Description Framework)/RDFS (Resource Description Framework Schema)* [9, 4] syntax. Three OWL species of increasing descriptive power have been specified: (a) *OWL-Lite*, which is intended for lightweight reasoning but has limited expressive power; (b) *OWL-DL*, which provides the description logics expressivity and guarantees computational completeness and decidability of reasoning; and (c) *OWL-Full*, which has more flexible syntax than OWL-DL, but does not guarantee computational completeness and decidability of reasoning.

The basic functionality provided by OWL is: *(a) Import of XML Schema Datatypes* that extend or restrict the basic datatypes (e.g. ranges etc.). The imported datatypes have to be declared (using the rdfs:Datatype construct), as *RDFS datatypes*, in the ontologies they are used; *(b) Definition of OWL Classes* (using the owl:Class construct), organized in subclass hierarchies (using the rdfs:subClassOf construct), for the representation of sets of individuals sharing some properties. Complex OWL classes can be defined via *set operators* (using the owl:intersectionOf, owl:unionOf and owl:complementOf constructs) or via *direct enumeration* of their members (using the owl:oneOf construct); *(c) Definition of OWL Individuals*, essentially instances of the OWL classes, following the restrictions imposed on the class in which they belong; and *(d) Definition of OWL Properties*, which may form property hierarchies (using the rdfs:subPropertyOf construct), for the representation of the features of the OWL class individuals. Two kinds of properties are provided by OWL: *(i) Object Properties*, defined using the owl:ObjectProperty construct, which relate individuals of one OWL class (the property domain, defined using the rdfs:domain construct) with individuals of another OWL class (the property range, defined using the rdfs:range construct); and *(ii) Datatype Properties*, defined using the owl:DatatypeProperty construct, which relate individuals belonging to one OWL class (the property domain) with values of a given datatype (the property range). Restrictions may be defined on OWL class properties (using the owl:Restriction construct), including type (using the owl:allValuesFrom construct), cardinality (using the owl:minCardinality, owl:maxCardinality and owl:cardinality constructs), and value (using the owl:hasValue construct) restrictions. OWL classes, (object and datatype) properties and individuals are identified by unique identifiers, that are specified in the "rdf:ID" attribute. They may also have labels, defined using the rdfs:label construct, and textual descriptions, defined using the rdfs:comment construct.

3 XS2OWL Overview

We present in this section the XS2OWL model for transforming XML Schema constructs in OWL-DL and its realization in the XS2OWL system. The XS2OWL system transforms every XML Schema it takes as input, through the implementation of the XS2OWL transformation model (outlined in Fig. 1), into: *(a)* A *main* OWL-DL ontology that directly captures the XML Schema semantics in OWL-DL; *(b)* A *mapping* OWL-DL ontology that keeps the mapping of the rdf:IDs of the OWL constructs of the main ontology with the names of the XML Schema constructs and systematically captures the semantics of the XML Schema constructs that cannot be directly captured in the main ontology, since they cannot be represented by corresponding OWL constructs; and *(c)* A *datatypes* XML Schema that contains the simple XML Schema datatypes defined in the source XML Schema that are imported in the main ontology.

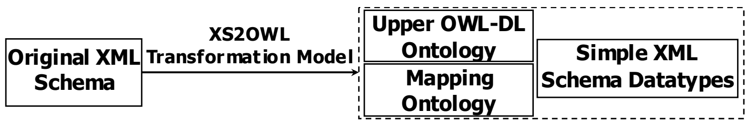


Fig. 1. Outline of the XS2OWL Transformation Model

The main OWL ontology essentially contains the OWL constructs to which the corresponding XML Schema constructs are transformed. As already mentioned, some of the XML Schema construct semantics cannot be expressed in OWL. The semantics of these constructs do not affect the domain ontologies that may extend the main ontology and they are not used by the OWL reasoners; however, they are important when individuals extending the main ontology have to be transformed back to valid XML descriptions compliant with the source XML Schema. For example, the elements of an XML Schema sequence should appear in a predefined order, while the OWL properties that are the constructs corresponding to the elements are always organized in unordered sets. As a consequence, the ordering information cannot be directly captured in the main ontology.

In order to support this functionality, we have defined a model that allows transforming the OWL constructs back to XML Schema constructs. This model captures the semantics of any arbitrary XML schema that cannot be represented in OWL and is expressed as an OWL-DL ontology, the *OWL2XMLRules Ontology* (available at http://www.music.tuc.gr/ontologies/OWL2XMLRules/OWL2XMLRules). For a particular XML Schema that is being transformed to OWL-DL, XS2OWL generates a mapping ontology, which extends the OWL2XMLRules ontology with individuals, keeps the mapping of the rdf:IDs of the OWL constructs of the main ontology with the names of the XML Schema constructs and represents the constructs of the original schema that cannot be directly represented in OWL.

The classes of the OWL2XMLRules ontology that represent the semantics of the XML Schema constructs that cannot be directly mapped to OWL constructs during the XML Schema to OWL transformation are the following:

- The *DatatypePropertyInfoType* class, which captures information about the datatype properties that cannot be directly expressed in OWL.
- The *ElementInfoType* class, which captures information about the XML Schema elements that cannot be directly expressed in OWL.
- The *ComplexTypeInfoType* class, which captures information about the complex XML Schema types that cannot be directly expressed in OWL.
- The *ChoiceInfoType* and *SequenceInfoType* classes, which capture, respectively, information about the XML Schema choices and sequences that cannot be directly expressed in OWL.

This way, there is no information loss during the XS2OWL transformations of the XML Schema constructs to OWL-DL constructs. As a consequence, all the semantics of the XML Schemas can be utilized by OWL-based tools and applications. For example, consider the transformation of individuals formed according to the main ontologies to XML documents obeying the original XML Schemas. Since the semantics of the XML Schema constructs that cannot be directly expressed in OWL are captured in the mapping ontologies, such information (e.g. the sequence element order) will be used in order to guarantee that the produced documents will be valid.

The XS2OWL transformation model has been implemented using the *XML Stylesheet Transformation Language (XSLT)* [8].

4 The XS2OWL Transformation Model

We present in this section the XS2OWL model, which allows the transformation of the XML Schema constructs to OWL-DL constructs. An overview of the XS2OWL Transformation Model is provided in Table 1, while the transformations of the individual constructs are formally presented in the following paragraphs.

The XML Schema constructs are provided in the first column of Table 1, while the OWL constructs that represent them in the main ontology are provided in the second column. As shown in Table 1, the complex XML Schema types are mapped to OWL classes, since they both represent sets of entities with common features. The simple XML Schema datatypes are mapped to datatype declarations, since OWL does not directly support the definition of simple datatypes, but only allows using simple XML Schema datatypes that have been declared in the OWL ontologies. The attributes are mapped to datatype properties, since they both represent simple type features, while the (simple and complex type) elements are mapped to (datatype and object) properties. The sequences and the choices are represented by OWL unnamed classes formed using set operators and cardinality restrictions on the sequence/choice items. Finally, the annotations of the XML Schema constructs are mapped to OWL comments.

The mapping ontology constructs representing the semantics of the XML Schema constructs that cannot be expressed directly in OWL are presented in the third column and in the fourth column are shown the contents of the datatypes XML Schema.

Table 1. Overview of the XS2OWL Transformation Model

XML Schema Construct	**OWL-DL Representation**		
	Main Ontology	**Mapping Ontology**	**Datatypes**
Complex Type	Class	*ComplexTypeInfoType* individual	
Simple Datatype	Datatype Declaration		Simple Type
Element	(Datatype or Object) Property	*ElementInfoType* individual	
Attribute	Datatype Property	*DatatypePropertyInfoType* individual	
Sequence	Unnamed Class - Intersection	*SequenceInfoType* individual	
Choice	Unnamed Class - Union	*ChoiceInfoType* individual	
Annotation	Comment		

Complex XML Schema Type Transformation. Let the complex XML Schema type *ct*, which is formally described in (1), where: (a) *name* is the name of *ct*; (b) *cid* is the (optional) identifier of *ct*; (c) *base* is the (simple or complex) type extended by *ct*; (d) *attributes* is the list of the attributes of *ct*; (e) *sequences* is the list of the sequences of *ct*; and (f) *choices* is the list of the choices of *ct*.

$$ct(name, cid, base, attributes, sequences, choices) \qquad (1)$$

The XS2OWL transformation of *ct* is different, depending on the type extended by *ct* (if it is simple or complex). The attributes and the elements that are defined or referenced in *ct* are transformed, in both cases, into properties.

If *ct* extends a complex type, it is represented in the main ontology by the OWL class *c*, formally described in (2), where:

c(id, super_class, label, value_restrictions, cardinality_restrictions) (2)

- *id* is the unique rdf:ID of *c* and has *name* as value if *ct* is a top-level complex type. If *ct* is a complex type nested within the definition of an element *e*, *id* is a unique, automatically generated name of the form *concatenate(ct_name, '_', name, '_UNType')*, where *ct_name* is the name of the complex type containing *e*. If *e* is a top-level element, *ct_name* has the 'NS' string as value. The *concatenate(…)* algorithm takes as input an arbitrary number of strings and returns their concatenation;
- *super_class* states which class is extended by *ct* and has *base* as value;
- *label* is the label of *ct* and has *name* as value;
- *value_restrictions* is the set of the value restrictions that represent the fixed values that may exist for some *ct* attributes and *ct* sequence/choice elements. The value of the restrictions is the value of the "fixed" attribute of the *ct* attributes and the *ct* sequence/choice elements;
- *cardinality_restrictions* is the set of the cardinality restrictions assigned to the properties representing the *ct* attributes and the *ct* sequence/choice elements. The cardinality restrictions are generated as follows:
  - According to the value of the "use" attribute of the XML Schema attributes: (a) If "use" has the "required" value, a cardinality restriction of value 1 is generated; (b) If "use" has the "prohibited" value, a cardinality restriction of value 0 is generated; and (c) If "use" is absent or has the "optional" value, a maximum cardinality restriction of value 1 is generated;
  - Cardinality, minimum and maximum cardinality restrictions are generated for the elements of the complex type, according to the "minOccurs" and "maxOccurs" attribute values of the elements and/or the sequences, choices and model groups the elements are organized in.

The semantics of *ct* that cannot be represented in OWL are represented in the mapping ontology by the *ComplexTypeInfoType* individual *ct* that is formally described in (3), where: (a) *id* is the unique rdf:ID of *ct* and has *name* as value; (b) *type_id* represents the identifier of the OWL class *c* that represents *ct* in the main ontology. *type_id* is represented as the "typeID" datatype property of *ct*; (c) *dpi_list* is the list of the representations of the datatype properties of *c*; and (d) *container_list* is the list of the representations of the *ct* containers (sequences and/or choices).

ct(id, type_id, dpi_list, container_list) (3)

If *ct* extends a simple type, it is represented in the main ontology by the OWL class *c*, formally described in (4).

c(id, label, value_restrictions, cardinality_restrictions) (4)

The fact that *ct* extends the simple type *base* is represented in the main ontology by the datatype property *ep* that is formally described in (5), where: (a) *eid* is the unique rdf:ID of *ep* and has *concatenate(base, '_content')* as value; (b) *range* is the range of *ep* and has *base* as value; and (c) *domain* is the domain of *ep* and has the *id* of *c* as value.

$$ep(eid, erange, edomain) \quad (5)$$

The semantics of *ct* that cannot be represented in OWL are represented in the mapping ontology by the *ComplexTypeInfoType* individual *ct* that corresponds to *c* and is formally described in (3), and the *DatatypePropertyInfoType* individual *dpi* that corresponds to *ep* and is formally described in (6). *dpi* states that the *ep* represents the fact that *ct* extends the simple type *base* through the 'Extension' value of the *dpi_type*, represented by the "datatypePropertyType" datatype property.

$$dpi(id, did, dpi\_type) \quad (6)$$

As an example, consider the complex type "ct" that extends the string datatype and is shown in Fig. 2. The "ct" complex type is represented in the main ontology by the "ct" OWL class, shown in Fig. 3, together with the "content__xs_string" datatype property, which states that "ct" is an extension of xs:string. The information about "ct" in the mapping ontology is shown in Fig. 4.

```
<xs:complexType name="ct">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="a">
    <xs:simpleType>
     <xs:restriction base="xs:integer"/>
    </xs:simpleType>
   </xs:attribute>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

Fig. 2. Definition of the "ct" XML Schema simple datatype. "ct" extends the string datatype with the "a" attribute. "a" is of a simple anonymous type that is a restriction of integer.

```
<owl:Class rdf:ID="ct">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#a__ct_a_UNType"/>
   <owl:maxCardinality rdf:datatype="&xsd;integer">1</owl:maxCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#content__xs_string"/>
   <owl:cardinality rdf:datatype= "&xsd;integer">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:label>ct</rdfs:label>
</owl:Class>
<owl:DatatypeProperty rdf:ID="content__xs_string">
 <rdfs:domain rdf:resource="#ct"/>
 <rdfs:range rdf:resource="&xs;string"/>
</owl:DatatypeProperty>
```

Fig. 3. Class representing the "ct" complex type of Fig. 2 in the main ontology

```
<ox:ComplexTypeInfoType rdf:ID="ct">
 <ox:typeID>ct</ox:typeID>
 <ox:DatatypePropertyInfo>
  <ox:DatatypePropertyInfoType rdf:ID="ct_a__ct_a_UNType">
   <ox:datatypePropertyID>a__ct_a_UNType</ox:datatypePropertyID>
   <ox:XMLConstructID>a</ox:XMLConstructID>
   <ox:datatypePropertyType>Attribute</ox:datatypePropertyType>
  </ox:DatatypePropertyInfoType>
 </ox:DatatypePropertyInfo>
 <ox:DatatypePropertyInfo>
  <ox:DatatypePropertyInfoType rdf:ID="ct_content__xs_string">
   <ox:datatypePropertyID>content__xs_string</ox:datatypePropertyID>
   <ox:datatypePropertyType>Extension</ox:datatypePropertyType>
  </ox:DatatypePropertyInfoType>
 </ox:DatatypePropertyInfo>
</ox:ComplexTypeInfoType>
```

Fig. 4. The ComplexTypeInfoType and DatatypePropertyInfoType individuals generated in the mapping ontology for the complex type “ct”, shown in Fig. 2

Simple XML Schema Datatype Transformation. Let the simple XML Schema type *st*, formally described in (7), where *body* is the body of the definition of *st*, *id* is the identifier of *st* and *name* is the name of *st*.

$$st(name, id, body) \tag{7}$$

XS2OWL transforms *st* into the simple datatype *st’* in the *datatypes* XML Schema, formally described in (8), and the *dd* datatype declaration in the main ontology, formally described in (9).

$$st'(name', id, body) \tag{8}$$

$$dd(about, is\_defined\_by, label) \tag{9}$$

The *st'* simple datatype has the same *body* and *id* with *st*, while *name'* is formed as follows: If *st* is a top-level simple type, *name'* is the value of its “name” attribute. If *st* is a simple type nested in the *ae* XML Schema construct (that may be an attribute or an element), *name'* has the value *id* if the identifier of *st* is not null. If the identifier of *st* is null, *name'* has as value the result of *concatenate(ct_name, '_', ae_name, '_UNType')*, where *ae_name* is the name of the property that represents *ae* and *ct_name* is the name of the complex type containing *ae*. If *ae* is a top-level attribute or element, *ct_name* has the ‘NS’ string as value;

The *dd* datatype declaration carries the following semantics: (a) *about* is the URI of *st'* referenced by the datatype declaration and is of the form *concatenate(uri, name')*, where *uri* is the URI of the datatypes XML Schema; (b) *is_defined_by* specifies where the datatype definition is located and has the *url* value; and (c) *label* is the label of *dd* and has *name'* as value.

As an example, consider the nested simple datatype of Fig. 2, which is defined in the “a” attribute of the “ct” complex type. It is transformed to the top-level simple datatype shown in Fig. 5, and the OWL datatype declaration shown in Fig. 6.

```
<xs:simpleType name="ct_a_UNType">
 <xs:restriction base="xs:integer"/>
</xs:simpleType>
```

Fig. 5. Top-level datatype representing the nested datatype of Fig. 2 in the datatypes XML Schema

```
<rdfs:Datatype rdf:about="&datatypes;ct_a_UNType">
 <rdfs:isDefinedBy rdf:resource="&datatypes;"/>
 <rdfs:label>ct_a_UNType</rdfs:label>
</rdfs:Datatype>
```

Fig. 6. Declaration of the "ct_a_UNType" simple datatype of Fig. 5 in the main ontology

XML Schema Element Transformation. Let the XML Schema element *e*, formally described in (10), where *name* is the name of *e*, *eid* is the identifier of *e*, *type* is the type of *e*, *ct_name* is the name of the complex XML Schema type *c_type* in the context of which *e* is defined (if *e* is a top-level attribute, *ct_name* has the null value), *sub_group* is an (optional) element being extended by *e*, *fixed* is the (optional) fixed value of *e*, *default* is the (optional) default value of *e, min* is the minimum number of occurrences of *e*, *max* is the maximum number of occurrences of *e* and *pos* is the position of *e* if *e* is a sequence element.

$$e(name,\ type,\ eid,\ ct\_name,\ sub\_group,\ fixed,\ default,\ min,\ max,\ pos) \quad (10)$$

XS2OWL represents *e* in the main ontology as a (datatype if *e* is of simple type, object if *e* is of complex type) property *p*, formally described in (11), where: (a) *id* is the unique rdf:ID of *p* and has *concatenate(name, '__', type)* as value; (b) *range* is the range of *p* and has *type* as value; (c) *domain* is the domain of *p* and takes the value of *ct_name*; (d) *label* is the label of *p* and has *name* as value; and (e) *super_property* is the specification of the property specialized by *p* and has *sub_group* as value.

$$p(id,\ range,\ domain,\ label,\ super\_property) \quad (11)$$

In the mapping ontology *e* is represented by the *ElementInfoType* individual *ei*, formally described in (12), where: (a) *id* is the unique rdf:ID of *ei* and has *concatenate(ct_name, '_', name, '__', type)* as value; (b) *pid* is the rdf:ID of the *p* property that represents *e* in the main ontology. *pid* is represented by the "propertyID" datatype property of *dpi*; (c) *xml_name* is the name of *e* and has *name* as value. *xml_name* is represented by the "elementID" datatype property of *dpi*; (d) *def_val* represents the default value of *e* and has *default* as value. *def_val* is represented as the "defaultValue" datatype property of *dpi*; (e) *min_occ* represents the minimum number of occurrences of *e* and has *min* as value. *min_occ* is represented by the "minOccurs" datatype property of *e*; (f) *max_occ* represents the maximum number of occurrences of *e* and has *max* as value. *max_occ* is represented by the "maxOccurs" datatype property of *e*; and (g) *position* represents the position of *e* if *e* is a sequence element. *position* is represented by the "elementPosition" datatype property of *e*.

$$ei(id,\ pid,\ xml\_name,\ def\_val,\ min\_occ,\ max\_occ,\ position) \quad (12)$$

In addition, if *e* is of simple type, a *DatatypePropertyInfoType* individual *dpi*, formally described in (13), is generated in the mapping ontology, where: (a) *id* is the unique rdf:ID of *dpi* and has *concatenate(ct_name, '_', name, '__', type)* as value; (b) *did* is the rdf:ID of the *p* datatype property that represents *e* in the main ontology. *did* is represented by the "datatypePropertyID" datatype property of *dpi*; (c) *xml_name* is the name of *e* and has *name* as value. *xml_name* is represented by the "XMLConstructID" datatype property of *dpi*; (d) *dpi_type* represents the construct which has been mapped to *p* and has the value *'Element'*; and (e) *def_val* represents the default value of *e* and has *default* as value.

$$dpi(id, did, xml\_name, dpi\_type, def\_val) \tag{13}$$

As an example, consider the "e1" element, shown in Fig. 7, of type "c_type2", which is defined in the context of the complex type "c_type1". The "e1" element is transformed to the OWL object property "e1__c_type2" of the main ontology (shown in Fig. 8) and the *ElementInfoType* individual "c_type1_e1__c_type2__ei" of the mapping ontology (shown in Fig. 9).

```
<xs:element name="e1" type="c_type2"/>
```

Fig. 7. XML Schema definition of the "e1" element, nested in the complex type "c_type1"

```
<owl:ObjectProperty rdf:ID="e1__c_type2">
  <rdfs:domain rdf:resource="#c_type1"/>
  <rdfs:range rdf:resource="#c_type2"/>
  <rdfs:label>e1</rdfs:label>
</owl:ObjectProperty>
```

Fig. 8. The object property representing the "e1" element of Fig. 7 in the main ontology

```
<ox:ElementInfoType rdf:ID="c_type1_e1__c_type2__ei">
  <ox:propertyID>e1__c_type2</ox:propertyID>
  <ox:elementID>e1</ox:elementID>
</ox:ElementInfoType>
```

Fig. 9. *ElementInfoType* individual representing the element of Fig. 7 in the mapping ontology

The XML Schema model groups essentially are sets of elements organized into (possibly nested) lists and choices. Let the XML Schema model group *g*, formally described in (14), which is comprised of n sequences/choices, where *g_name* is the model group name, *g_id* is the model group identifier and l_i with 1≤i≤n are the group sequences/choices.

$$g(g\_name, g\_id, (l_1 , \ldots, l_n)) \tag{14}$$

If l_i is a sequence/choice of m elements formally described in (15), then for 1≤j≤m, $name_{ij}$, is the name of e_{ij}, eid_{ij}, is the identifier of e_{ij}, $type_{ij}$ is the value of the "type" attribute of e_{ij} and sub_group_{ij} is an element being extended by e_{ij}.

$$l_i(lid_i, e_{i1}(name_{i1}, eid_{i1}, type_{i1}, sub\_group_{i1}), \ldots, e_{im}(name_{im}, eid_{im}, type_{im}, sub\_group_{im})) \tag{15}$$

XS2OWL represents *g* in the main ontology as the (datatype or object) properties p_{ij}, formally described in (16), where: (a) id_{ij} is the unique rdf:ID of p_{ij} and has *concatenate(g_name, '__', name$_{ij}$)* as value; (b) *range$_{ij}$* is the range of *p$_{ij}$* and has *type$_{ij}$* as value; (c) *label$_{ij}$* is the label of *p$_{ij}$* and has *name$_{ij}$* as value; and (d) *super_property$_{ij}$* represents the property specialized by *p* and has *sub_group$_{ij}$* as value.

$$p_{ij}(id_{ij},\ range_{ij},\ label_{ij},\ super\_property_{ij}) \tag{16}$$

In the mapping ontology *g* is represented by an *ElementInfoType* individual *ei* for each element *e$_{ij}$*, formally described in (12). If *e$_{ij}$* is represented in the main ontology by a datatype property, a *DatatypePropertyInfoType* individual *dpi*, formally described in (13), is also generated in the mapping ontology for the *e$_{ij}$*.

XML Schema Attribute Transformation. Let the XML Schema attribute *a*, formally described in (17), where *name* is the name of *a*, *aid* is the identifier of *a*, *type* is the type of *a*, *ct_name* is the name of the complex XML Schema type *c_type* in the context of which *a* is defined (if *a* is a top-level attribute, *ct_name* has the null value), *fixed* is the fixed value of *a* and *default* is the default value of *a*.

$$a(name,\ aid,\ type,\ ct\_name,\ fixed,\ default) \tag{17}$$

XS2OWL represents *a* in the main ontology as an OWL datatype property *dp*, formally described in (18), where: (a) *id* is the unique rdf:ID of *dp* and has *concatenate(name, '__', type)* as value; (b) *range* is the range of *dp* and has *type* as value; (c) *domain* is the domain of *dp* and takes the value of *ct_name*; and (d) *label* is the label of *dp* and has *name* as value.

$$dp(id,\ range,\ domain,\ label,\ comment) \tag{18}$$

In the mapping ontology *a* is represented as a *DatatypePropertyInfoType* individual *dpi*, formally described in (13), where *dpi_type* represents the construct which has been mapped to *dp* and has the value *'Attribute'*.

As an example, consider the "a" attribute of Fig. 2, which is transformed to the datatype property of the main ontology shown in Fig. 10 and the *DatatypePropertyInfoType* individual of the mapping ontology shown in Fig. 11.

```
<owl:DatatypeProperty rdf:ID="a__ct_a_UNType">
 <rdfs:domain rdf:resource="#ct"/>
 <rdfs:range rdf:resource="&datatypes;ct_a_UNType"/>
 <rdfs:label>a</rdfs:label>
</owl:DatatypeProperty>
```

Fig. 10. The datatype property representing the "a" attribute of Fig. 2 in the main ontology

```
<ox:DatatypePropertyInfo>
 <ox:DatatypePropertyInfoType rdf:ID="ct_a__ct_a_UNType">
  <ox:datatypePropertyID>a__ct_a_UNType</ox:datatypePropertyID>
  <ox:XMLConstructID>a</ox:XMLConstructID>
  <ox:datatypePropertyType>Attribute</ox:datatypePropertyType>
 </ox:DatatypePropertyInfoType>
</ox:DatatypePropertyInfo>
```

Fig. 11. The DatatypePropertyInfoType individual representing the "a" attribute of Fig. 2 in the mapping ontology

```
algorithm sequence_restr(sequence, max_p, min_p)
minOc=sequence/@minOccurs
maxOc=sequence/@maxOccurs
temp=''
if ((minOc*min_p=0) and (maxOc='unbounded' or max_p='unbounded'))
 return ''
else
 for each sequence item
  if the item is element
   if (maxOc='unbounded' or max_p='unbounded')
    temp=seq_element_restr(item, min_p*minOc, 'unbounded')
   else
    temp=seq_element_restr(item, min_p*minOc, max_p*maxOc)
   end if
  else if the item is sequence
   if (maxOc='unbounded' or max_p='unbounded')
    temp=sequence_restr(item, min_p*minOc, 'unbounded')
   else
    temp=sequence_restr(item, min_p*minOc, max_p*maxOc)
   end if
  else
   if (maxOc='unbounded' or max_p='unbounded')
    temp=choice_restr(item, min_p*minOc, 'unbounded')
   else
    temp=choice_restr(item, min_p*minOc, max_p*maxOc)
   end if
  end if
  ret=concatenate(ret, temp)
 end for
 return intersection(ret)
end if
end algorithm

algorithm seq_element_restr(element, max_p, min_p)
minOc=sequence/@minOccurs
maxOc=sequence/@maxOccurs
if ((minOc*min_p=0) and (maxOc='unbounded' or max_p='unbounded'))
 return ''
else if (maxOc='unbounded' or max_p='unbounded')
 return min_car(item, minOc*min_p)
else if (maxOc*max_p=minOc*min_p)
 return cardinality(item, minOc*min_p)
else
 return intersection(min_car(item, minOc*min_p), max_car(item, maxOc*max_p))
end if
end algorithm
```

Fig. 12. Algorithm for the generation of unnamed OWL classes representing XML Schema Sequences, where: (a) The min_car(item,min) algorithm produces an owl:minCardinality restriction on the property that represents "item" with value "min"; (b) The cardinality(item,val) algorithm produces an owl:cardinality restriction on the property that represents "item" with value "val"; and (c) The max_car(item,max) algorithm produces an owl:maxCardinality restriction on the property that represents "item" with value "max".

XS2OWL transforms the XML Schema attribute groups into sets of datatype properties, each of which represents an attribute that belongs to the attribute group. Let *ag* be an XML Schema attribute group comprised of n attributes, formally described in (19), where *ag_name* is the attribute group name, *ag_ id* is the attribute group identifier and a_i, with 1≤i≤n are the group attributes, formally described in (20). $name_i$ is the name of a_i, aid_i is the identifier of a_i and $type_i$ is the type of a_i.

$$ag(ag\_name, ag\_id, (a_1, \ldots, a_n)) \qquad (19)$$

$$a_i(name_i, aid_i, type_i) \quad (20)$$

XS2OWL represents *ag* in the main ontology as a set of datatype properties dp_i, $1 \le i \le n$, formally described in (21), where: (a) id_i is the unique rdf:ID of dp_i and has *concatenate(ag_name, '__', name*$_i$*)* as value; (b) $range_i$ is the range of dp_i and has $type_i$ as value; and (c) $label_i$ is the label of dp_i and has $name_i$ as value.

$$dpi_i(id_i, range_i, label_i) \quad (21)$$

In the mapping ontology *ag* is represented by a *DatatypePropertyInfoType* individual *dpi* for each attribute, formally described in (13), with *dpi_type* having the value *'Attribute'*.

XML Schema Sequence and Choice Transformation. XS2OWL transforms both the sequences and the choices into OWL-DL unnamed classes formed using set operators and cardinality restrictions on the sequence/choice items. The classes that represent the complex types where the sequences/choices are defined or referenced are subclasses of the unnamed OWL classes that represent the sequences/choices. The sequence and the choice item cardinality must always be a multiple of an integer in the range [*i_min_occurs*, *i_max_occurs*], where *i_min_occurs* and *i_max_occurs* are the values of the "minOccurs" and the "maxOccurs" attributes of the item.

The sequences are represented in the main ontology as unnamed classes, formed from the intersection of the cardinality restrictions of the sequence items. The algorithm that transforms the XML Schema sequences to unnamed OWL classes is shown in Fig. 12. For the transformation of a sequence *s* the algorithm is initially called as *sequence_restr(s, 1, 1)*.

As an example, consider the sequence shown in Fig. 13, which is defined in the context of a complex type *c*. The sequence is represented in the main ontology by the unnamed OWL class shown in Fig. 14, of which the class that represents the complex type *c* is a subclass.

```
<xs:sequence minOccurs="2" maxOccurs="2">
 <xs:element name="e1" type="xs:string"/>
 <xs:element name="e2" type="xs:string" maxOccurs="3"/>
</xs:sequence>
```

Fig. 13. XML Schema Sequence defined in the context of the complex type *c*

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Restriction>
      <owl:onProperty rdf:resource="#e1__xs_string"/>
      <owl:cardinality rdf:datatype="&xsd;integer">2</owl:cardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#e2__xs_string"/>
      <owl:minCardinality rdf:datatype="&xsd;integer">2</owl:minCardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#e2__xs_string"/>
      <owl:maxCardinality rdf:datatype="&xsd;integer">6</owl:maxCardinality>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Fig. 14. Representation of the sequence of Fig. 13 in the main ontology

The choices are represented in the main ontology as unnamed classes, formed from the union of the allowed combinations of the choice elements. The algorithm that transforms the XML Schema choices to unnamed OWL classes is shown in Fig. 15.

```
algorithm choice_restr(choice, maxOc, minOc)
ret=''
if (minOc=0 and maxOc='unbounded'), return ret
else if maxOc='unbounded'
 nz=number of choice items with minOccurs<>0
 z=number of choice items with minOccurs=0
 nzi=choice items with minOccurs<>0
 mat=distribute(nz,0,minOc)
 for i=1 to mat rows
  for j=1 to nz
   min=nzi[j]/@minOccurs
   if nzi[j] is element, add min_car(nzi[j], min*mat[i,j]) to ret
   else if nzi[j] is sequence, add sequence_restr(nzi[j], min*mat[i,j],
'unbounded') to ret
   else add choice_restr(nzi[j],min*mat[i,j],'unbounded') to ret
   end if
  end for
  ret=intersection(ret)
 end for
 if z>0
  z_temp=intersection of deep_cardinality(nzi[i],0) (i from 1 to nz)
  return union(z_temp, ret)
 else return ret
 end if
else
 nzu=number of choice items with (minOccurs<>0 and maxOccurs<>'unbounded')
 zu=number of choice items with (minOccurs=0 and maxOccurs='unbounded')
 nzui=choice items with (minOccurs<>0 and maxOccurs<>'unbounded')
 mat=distribute(nz,minOc,maxOc)
 for i=1 to mat rows
  for j=1 to nz
   min=nzui[j]/@minOccurs
   max=nzui[j]/@minOccurs
   if nzui[j] is element
    if (max='unbounded' or maxOc='unbounded'), add min_car(nzui[j],
min*mat[i,j]) to ret
    else if (min*minOc=0), add max_car(nzui[j], max*mat[i,j]) to ret
    else (if min*minOc=max*maxOc), add cardinality(nzui[j], max*mat[i,j])
to ret
    else add intersection(min_car(nzui[j], min*mat[i,j]), max_car(nzui[j],
max*mat[i,j])) to ret
    end if
   else if nzui[j] is sequence
    if (max='unbounded'), add sequence_restr(nzui[j], min*mat[i,j], 'un-
bounded') to ret
    else add sequence_restr(nzui[j], min*mat[i,j], max*mat[i,j]) to ret
    end if
   else
    if (max='unbounded'), add choice_restr(nzui[j], min*mat[i,j], 'un-
bounded') to ret
    else add choice_restr(nzui[j], min*mat[i,j], max*mat[i,j]) to ret
    end if
   end if
```

Fig. 15. Algorithm for the generation of unnamed OWL classes representing XML Schema Choices, where: (a) The deep_cardinality(item,val) algorithm produces an owl:cardinality restriction with value "val" on each property that represents "item" or a (sequence or choice) item of "item"; and (b) The distribute(item_num, min_val, max_val) algorithm calculates the allowed combination of the occurrences of "item_num" items so that the sum of their occurrences is between "min_val" and "max_val".

```
   end for
   intersection(ret)
  end for
  if zu>0
   zu_temp=intersection of cardinality(nzui[i],0) (i from 1 to nzu)
   return union(zu_temp, ret)
  else return ret
  end if
 end if
 end algorithm
```

Fig. 15. (continued)

As an example, consider the choice shown in Fig. 16. The choice is represented in the main ontology by the unnamed OWL class shown in Fig. 17.

```
<xs:choice minOccurs="0">
 <xs:element name="e2" type="xs:string" minOccurs="2" maxOccurs="2"/>
 <xs:element name="e3" type="xs:string" maxOccurs="2"/>
</xs:choice>
```

Fig. 16. XML Schema choice defined in the context of a complex type

```
 <owl:Class>
  <owl:unionOf rdf:parseType="Collection">
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
     <owl:Restriction>
      <owl:onProperty rdf:resource="#e2__xs_string"/>
      <owl:maxCardinality rdf:datatype="&xsd;integer">0</owl:maxCardinality>
     </owl:Restriction>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#e3__xs_string"/>
      <owl:maxCardinality rdf:datatype="&xsd;integer">2</owl:maxCardinality>
     </owl:Restriction>
    </owl:intersectionOf>
   </owl:Class>
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
     <owl:Restriction>
      <owl:onProperty rdf:resource="#e2__xs_string"/>
      <owl:maxCardinality rdf:datatype="&xsd;integer">2</owl:maxCardinality>
     </owl:Restriction>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#e3__xs_string"/>
      <owl:maxCardinality rdf:datatype="&xsd;integer">0</owl:maxCardinality>
     </owl:Restriction>
    </owl:intersectionOf>
   </owl:Class>
  </owl:unionOf>
 </owl:Class>
```

Fig. 17. Representation of the XML Schema Choice of Fig. 16 in the main ontology

If the maximum number of occurrences of a sequence/choice has a large value (but is not unbounded), the manual generation of the unnamed classes is tedious and time-consuming and thus becomes error-prone and practically impossible.

Notice that the exact sequence/choice cardinalities cannot be computed when a choice item is contained in a sequence/choice with unbounded maximum number of occurrences. In addition, information regarding the sequence element ordering cannot be represented in OWL. This information is captured in the mapping ontology. Let *sc* be a sequence or choice formally described in (22), where *sc_id* is the identifier of *sc*, *c_type* is the complex type in which *sc* is defined or referenced, *min* is the minimum number of occurrences of *sc*, *max* is the maximum number of occurrences of *sc* and *elements* is the list of the elements of *sc*.

$$sc(sc\_id,\ c\_type,\ min,\ max,\ elements) \tag{22}$$

XS2OWL represents *sc* in the mapping ontology by the (*SequenceInfoType* if *sc* is a sequence, *ChoiceInfoType* if *sc* is a choice) individual *st* formally described in (23), where: (a) *id* is the unique rdf:ID of *st* and has *concatenate(ct_name, '__', i)* as value, where *ct_name* is the name of the class that represents *c_type* in the main ontology and *i* is the index of *sc* in *c_type*; (b) *min_occ* represents the minimum number of occurrences of *sc* and has *min* as value; (c) *max_occ* represents the maximum number of occurrences of *sc* and has *max* as value; and (d) *e_rep* is the list of the representations of the *elements* of *sc*.

$$st(id,\ min\_occ,\ max\_occ,\ e\_rep) \tag{23}$$

5 Evaluation of the XS2OWL Model

The XS2OWL model and its implementation have been applied to several well-accepted standards. We present here the results of the XS2OWL evaluation.

In order to acquire extensive empirical evidence, we applied XS2OWL to several very large and well-accepted standards expressed in XML Schema: The MPEG-7 Multimedia Description Schemes (MDS) and the MPEG-21 Digital Item Adaptation (DIA) Architecture in the multimedia domain, the IEEE LOM and the SCORM in the e-learning domain and the METS standard for Digital Libraries. The result was the transformation of the XML Schema constructs to OWL for each one of those standards. We then enriched the OWL specifications with OWL domain ontologies and produced individuals following the ontologies. Finally, we converted the individuals to XML syntax, valid with respect to the original XML Schemas. The transformations were successful for these standards due to the utilization of the mapping ontologies. We have also found that in all cases the semantics of the standards were fully captured in the main and mapping ontologies generated by the XS2OWL system.

Then we looked closely the formal semantics of the generated OWL ontologies. Since we had in our previous research efforts [17] manually transformed the MPEG-7 MDS and the MPEG-21 DIA Architecture in OWL-DL, we compared the semantics captured in the manually defined ontologies with the semantics captured in the automatically generated ones. The comparison has shown that all the semantics captured during the manual transformations were also captured during the automatic transformations. Then, we compared the manually produced mapping ontology for the Semantic DS of the MPEG-7 MDS with the corresponding part of the automatically produced mapping ontology for the MPEG-7 MDS. Again, the comparison has shown that all the semantics captured in the manually created ontology are also accurately captured in the automatically produced one.

6 Conclusions – Future Work

We have presented in this paper the XS2OWL model that allows the automatic transformation of XML Schemas into OWL-DL ontologies. This transformation allows domain ontologies in OWL to be integrated and logic-based reasoners to be used for various applications, as for example for knowledge extraction from multimedia data. XS2OWL allows the conversion of the generated OWL information back to XML. We have presented also a system that implements the XS2OWL model. We have used the implemented system to validate our approach with a number of well-accepted and extensive standards expressed in XML Schema. The automatically created ontologies have been found to accurately capture the semantics of the XML Schemas.

Our future work in this area includes experimentation that will be conducted in order to evaluate the enhancement of the retrieval effectiveness that will be achieved through the utilization of the products of the XS2OWL model. In particular, we will pose the same queries on top of XML repositories containing (a) Standard-based XML descriptions that were created from scratch; and (b) Standard-based XML descriptions that were derived from the transformation of OWL constructs produced after the semantic processing of OWL individuals defined based on the OWL ontologies produced from the application of the XS2OWL methodology and software on the standards. Precision and recall will be calculated in both cases and will be compared.

Acknowledgments. The work presented here was partially funded in the scope of the DELOS II Network of Excellence in Digital Libraries (IST – Project Record #26059).

References

1. ADL Technical Team: Sharable Content Object Reference Model (SCORM) (2004)
2. An, Y., Borgida, A., Mylopoulos, J.: Constructing Complex Semantic Mappings Between XML Data and Ontologies. In: International Semantic Web Conference, pp. 6–20 (2005)
3. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., Cowan, J. (eds.): Extensible Markup Language (XML) 1.1. W3C Recommendation (2006), http://www.w3.org/TR/xml11/
4. Brickley, D., Guha, R.V. (eds.): RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation (2004), http://www.w3.org/TR/rdf-schema
5. Chang, S.F., Sikora, T., Puri, A.: Overview of the MPEG-7 standard. IEEE Transactions on Circuits and Systems for Video Technology 11, 688–695 (2001)
6. Fallside, D., Walmsley, P. (eds.): XML Schema Part 0: Primer. W3C Recommendation (2001), http://www.w3.org/TR/xmlschema-0/
7. García, R., Celma, O.: Semantic Integration and Retrieval of Multimedia Metadata. In: proceedings of the Semannot 2005 Workshop (2005)
8. Kay, M. (ed.): XSL Transformations (XSLT) Version 2.0. W3C Recommendation (2007), http://www.w3.org/TR/xslt20/
9. Manola, F., Milles, E. (eds.): RDF Primer. W3C Recommendation (2004), http://www.w3.org/TR/rdf-primer
10. METS: Metadata Encoding and Transmission Standard (METS) Official Website http://www.loc.gov/standards/mets/

11. IEEE LTSC 2002: IEEE 1484.12.1-2002 – Learning Object Metadata Standard. http://ltsc.ieee.org/wg12/
12. ISO/IEC: 21000-7:2004 – Information Technology – Multimedia Framework (MPEG-21) – Part 7: Digital Item Adaptation (2004)
13. ISO/IEC: 15938-5:2003 – Information Technology –Multimedia content description interface – Part 5: Multimedia description schemes. First Edition, ISO/MPEG N5845 (2003)
14. McGuinness, D.L., van Harmelen, F. (eds.): OWL Web Ontology Language: Overview. W3C Recommendation (2004), http://www.w3.org/TR/owl-features
15. Pereira, F.: The MPEG-21 standard: Why an open multimedia framework? In: Shepherd, D., Finney, J., Mathy, L., Race, N.J.P. (eds.) IDMS 2001. LNCS, vol. 2158, pp. 219–220. Springer, Heidelberg (2001)
16. Tsinaraki, C., Polydoros, P., Christodoulakis, S.: Interoperability support for Ontology-based Video Retrieval Applications. In: Enser, P.G.B., Kompatsiaris, Y., O'Connor, N.E., Smeaton, A.F., Smeulders, A.W.M. (eds.) CIVR 2004. LNCS, vol. 3115, pp. 582–591. Springer, Heidelberg (2004)
17. Tsinaraki, C., Polydoros, P., Christodoulakis, S.: Interoperability support between MPEG-7/21 and OWL in DS-MIRF. Transactions on Knowledge and Data Engineering (TKDE), Special Issue on the Semantic Web Era, pp. 219–232 (2007)
18. Tsinaraki, C., Polydoros, P., Kazasis, F., Christodoulakis, S.: Ontology-based Semantic Indexing for MPEG-7 and TV-Anytime Audiovisual Content. Multimedia Tools and Application Journal (MTAP) 26, 299–325 (2005)

Query Expansion and Interpretation to Go Beyond Semantic P2P Interoperability

Anthony Ventresque[1], Sylvie Cazalens[1],
Philippe Lamarre[1], and Patrick Valduriez[2]

[1] LINA, University of Nantes
FirstName.LastName@univ-nantes.fr
[2] INRIA and LINA, University of Nantes
Patrick.Valduriez@inria.fr

Abstract. In P2P data management systems, semantic interoperability between any two peers that do not share the same ontology relyes on ontology matching. The established correspondences, i.e. the "shared" parts of the ontologies are indeed essential to exchange information. But to what extent the "unshared" part can contribute to information exchange. In this paper, we address this question. We focus on a P2P document management system, where documents and queries are represented by semantic vectors. We propose a specific query expansion step at the query initiator's side and a query interpretation step at the document provider's. Through these steps, unshared concepts contribute to evaluate the relevance of documents wrt a given query. The experiments show that the proposed method enables to correctly evaluate the relevance of a document even if concepts of a query are not shared. In some cases, we are able to find up to 90% of the documents that would be selected when all the concepts are shared.

1 Introduction

In peer-to-peer (P2P) data systems, semantic interoperability means that any two peers are able to exchange information of which meaning is correctly interpreted by both of them. Several solution in P2P data management use local mappings to ensure the systems global interoperability [5,8]. Most of the solutions focus on what (i.e. the concepts and relations) the peers share, which is important. However, no matter how the shared part is obtained (through consensus or mapping), there might be concepts (and relations) that are not consensual, and thus not shared but still useful for information exchange. Thus the question is to know whether the unshared parts are useful for information exchange.

In this paper, we restrict this question to the case of a P2P document management system, with unstructured or semi-structured documents. More precisely, we focus on semantic interoperability and information exchange between two peers, a query initiator p_1 and a document provider p_2, which use different ontologies but share some common concepts. Each of them represents its queries and documents, according to its own ontology. The, the problem we address is to *find documents which are relevant to a given query although the documents and the query may be both represented with concepts that are not all shared.*

We represent documents and queries by *semantic vectors* [11] which are a common way to represent unstructured documents. The principle is simple: each concept of the ontology is weighted according to its representiveness of the document. The same is done for the query. The resulting vector represents the document (respectively, the query) in the n-dimensional space formed by the n concepts of the ontology. Then the relevance of a document with respect to a query corresponds to the proximity of the vectors in the space.

In order to improve information exchange beyond the "shared part" of the ontology, we promote both query expansion (at the query initiator's side) and query interpretation (at the document provider's side). Query expansion may contribute to weight linked shared concepts, thus improving the document provider's understanding of the query. Similarly, by interpreting an expanded query with respect to its own ontology (i.e. by weighting additional concepts of its own ontology), the document provider may find additional related documents that would not be found by only using the matching concepts in the query and the documents. To our knowledge, the best of the problem of improving information exchange by using the unshared concepts of different ontology has not been addressed before. Our proposal is a first, encouraging solution.

This paper is organised as follows. Section 2 gives preliminary definitions. In Section 3, we present query expansion in the case of a shared ontology. Its main property is to keep separate the results of the propagation from each central concept of the query. Section 4 considers the case where two peers use different ontologies, and describes query interpretation. Section 5 gives preliminary experiments. Finally, we conclude in Section 6.

2 Preliminary Definitions

We use a semantic vector space, i.e. a multi-dimensional linear space with the concepts of an ontology as dimensions. The content of each document (respectively query) is abstracted to a semantic vector by characterizing it according to each concept. The more a given document is related to a given concept, the higher is the value of the concept in the semantic vector of the document.

We simply define an ontology as a set of concepts together with a set of relations between those concepts [4]. In our experiments we consider an ontology with only the is-a relation (specialization link). This does not restrict the generality of our relevance calculus.

Definition 1 (Semantic Vector). *A* semantic vector $\overrightarrow{v_\Omega}$*, or* $\overrightarrow{v}$ *when there is no ambiguity, is an application defined on the set of concepts* $\mathcal{C}_\Omega$ *of an ontology* Ω*:*

$\forall c \in \mathcal{C}_\Omega, \overrightarrow{v_\Omega} : c \rightarrow [0, 1]$

Expansion is based on weight propagation, which consists in weighting initially unweighted concepts which seem linked to weighted concepts. We propose to propagate weight from a given concept c_i , according to the similarity[7] of the other concepts with c_i . Thus, we introduce a similarity function sim_{c_i} which denotes the similarity to a *central concept* c_i .

Definition 2 (Similarity function). *Let c be a concept of $\mathcal{C}_\Omega$. Function sim_c: $\mathcal{C}_\Omega \rightarrow [0, 1]$, is a similarity function iff $sim_c(c) = 1$ and $0 \leq sim_c(c_j) < 1$ for all $c_j \neq c$ in $\mathcal{C}_\Omega$.*

The concepts of $\mathcal{C}_\Omega$ can then be ordered according to their similarity value. A propagation function $\mathcal{P}f_c$ is a decreasing function which assigns a weight to each similarity value, c being assigned the highest weight. Figure 1 is an example of a propagation function, inspired by membership functions used in fuzzy logic.

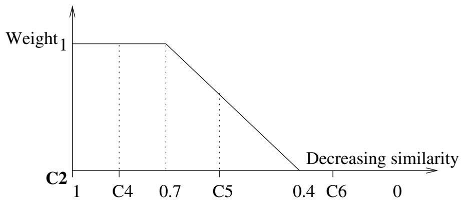


Fig. 1. Example of a $f_{0.7,0.4,1}$ function with central concept c_2

3 Query Expansion and Image Based Relevance

For the sake of simplicity, we assume in this section that the query initiator and the document provider use the same ontology but they can still differ on the similarity measures and the propagation functions.

Most propagation methods propagate the weight of each concept in *the same vector*. We call this kind of method "rough" propagation. Although the results are not bad, this method has some drawbacks, in particular, a possible unbalance of the relative importance of the initial concepts [6]. This is why we choose to keep separate the results of the propagation from different concepts in *semantic enriched dimensions* (SEDs).

First, let us denote by $\mathcal{C}_{\overrightarrow{q}}$ the set of the *central concepts* of query $\overrightarrow{q}$, i.e. those weighted concepts which best represent the query. Each central concept of $\mathcal{C}_{\overrightarrow{q}}$ is *semantically enriched* by propagation, in a separate vector (see figure 2).

Definition 3 (Query expansion). *Let $\overrightarrow{q}$ be a query vector; let c be a concept in $\mathcal{C}_{\overrightarrow{q}}$ and let $\mathcal{P}f_c$ be a propagation function.*

The semantically enriched dimension *of c, noted $\overrightarrow{sed}_c$, is the semantic vector defined by: $\forall c' \in \mathcal{C}_\Omega, \overrightarrow{sed}_c[c'] = \mathcal{P}f_c(c')$*

The expansion *of $\overrightarrow{q}$, noted $\mathcal{E}_{\overrightarrow{q}}$ is defined as: $\mathcal{E}_{\overrightarrow{q}} = \{\overrightarrow{sed}_c : c \in \mathcal{C}_{\overrightarrow{q}}\}$*

The relevance of a given document is computed using the cosine of its *image* wrt the query and the query itself. This image is obtained using the expansion of the

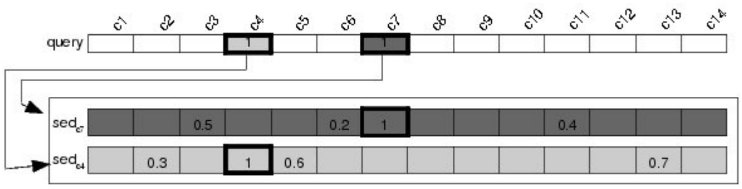


Fig. 2. A query expansion composed of 2 semantically enriched dimensions

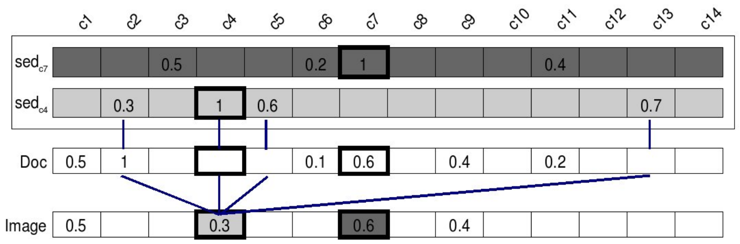


Fig. 3. Obtaining the image of a document

query (i.e. the set of SEDs). Given a SED $\overrightarrow{sed_c}$, we consider the product of the respective values of each concept in $\overrightarrow{sed_c}$ and $\overrightarrow{d}$. The image of $\overrightarrow{d}$ keeps track of the best value assigned to one of the linked concepts if it is better than $\overrightarrow{d}$ $[c]$, which is the initial value of c. All the central concepts of the initial query vector are then weighted in the image of the document as far as the document is related to them.

4 Relevance in the Context of Unshared Concepts

We now assume that the query initiator, p_1, and the document provider, p_2, use different ontologies, respectively noted Ω_1 and Ω_2. Each peer also has its own similarity and propagation functions. We also assume that the peers *share* some common concepts: each of them regularly (although may be not often) computes an ontology matching algorithm which provides a non-empty set of correspondences (equivalences) between those concepts [3]. For the sake of simplicity of notations, when there is an equivalence, we make no difference between the name of the given concept at p_1's, its name at p_2's, and the identifier of the correspondence, which all refer to the same concept.

4.1 Overview of the Relevance Calculus

The query initiator and the document provider do not use the same vector spaces. An additional step is needed in order to be able to evaluate relevance in a same and single space. We call it *interpretation* of the query. Thus, the different steps involved in the relevance calculus of some document $\overrightarrow{d}$ of p_2 wrt a given query $\overrightarrow{q}$ initiated by p_1 are the following.

Query expansion. It remains unchanged. Peer p_1 computes an *expansion* of its query, which results in a set of SEDs. Each SED is expressed on the set $\mathcal{C}_{\Omega_1}$, no matter the ontology used by p_2. Then, the expanded query is sent to p_2, together with the initial query.
Query interpretation. Query interpretation by p_2 provides a set of interpreted SEDs on the set $\mathcal{C}_{\Omega_2}$ and an interpreted query. Each SED of the expanded query is interpreted separately. Interpretation is composed of two problems:

– The first problem is to find a concept that corresponds to the central concept of the SED. This is difficult when the central concept is not shared. It might even lead p_2 to introduce "new" concepts. Because of space limitations, we do not detail this part. In the following, we assume that the corresponding concept belongs to $\mathcal{C}_{\Omega_2}$, even if the initial concept is not shared, and that it keeps the weight of this latter.
– The second problem is to attribute weights to unshared concepts of $\mathcal{C}_{\Omega_2}$ which are linked to the SED. This is detailed below.

Image of the document and cosine calculus. They remain unchanged. Provider p_2 computes the image of its document with respect to the interpreted SEDs and then, its cosine based relevance with respect to the interpreted query, no matter the ontology used by p_1. This is possible because the previous interpretation step makes both the image of the document and the interpreted query belong to space $\mathcal{C}_{\Omega_2}$.

4.2 Interpretation of a SED

In this section, we describe the interpretation process for a given SED (expressed on $\mathcal{C}_{\Omega_1}$), of which central concept is noted c. The concept corresponding to c in $\mathcal{C}_{\Omega_2}$ is noted $i_{\mathcal{E}_{\overrightarrow{q}}}(c)$ and is assigned the weight of c. Peer $p2$ ranks its own concepts in function of $sim_{i_{\mathcal{E}_{\overrightarrow{q}}}(c)}$. Among these concepts, some are shared and their initial SED has a given weight, which we preserve in the interpretation. The problem is how to weight the unshared concepts, given that some of them might be more similar to $i_{\mathcal{E}_{\overrightarrow{q}}}(c)$ than shared concepts. Figure 4 illustrates our general solution. Let us call f_i a piecewise affine function which defines a weight for each similarity value in $[0, 1]$. To guide the definition of f_i, we use the weights given by $\overrightarrow{sed_c}$ to the shared concepts ($c1$, $c2$, $c3$ in figure 4). However, there might be several shared concepts that have the same similarity value with respect to $i_{\mathcal{E}_{\overrightarrow{q}}}(c)$, but have a different weight according to $\overrightarrow{sed_c}$. We only require function f_i to assign one of (or a function of) these values to the given similarity value. For instance, it can be the minimum value. Given a function f_i that satisfyes this condition, the unshared concepts ($c4$, $c5$, $c6$ in the figure) are assigned the weight they obtain by function f_i. This is illustrated for c_5 by a dotted arrow.

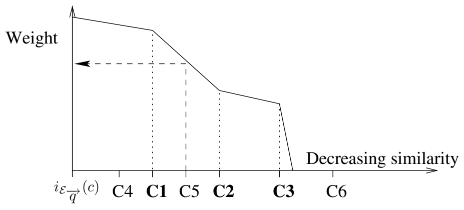


Fig. 4. SED interpretation: assigning weights to unshared non-central concepts

5 Preliminary Experiments

The ontology we use is lightweight, i.e. an ontology composed of a taxonomy of concepts and a taxonomy of relations : *WordNet*. We use the *Cranfield corpus*, a testing corpus consisting of 1400 documents and 225 queries in natural language, all related to aeronautical engineering[1]. For each query, each document is scored by humans as relevant or not relevant (boolean relevance). Although it is similar in size to other classical testing corpora like CACM, CISI, Medline, etc. it is small compared to recent TREC corpus. As we are not experts in textual IR nor in natural language processing, we do not focus on a large corpus for our experiments. However, this is one direction of our future work. *Semantic indexing* is the process which extracts concepts within documents or queries in natural language [9]. The aim is to find the most representative concepts for documents and queries. We use a program made in our lab : RIIO [2], which is based on the selection of synsets from WordNet. Thus, there is no human intervention in the process. We use a *similarity* function based on [1], because it has good properties and results. The *propagation* functions used are of the form $f_{1,l_2,v}$ (see figure 1)

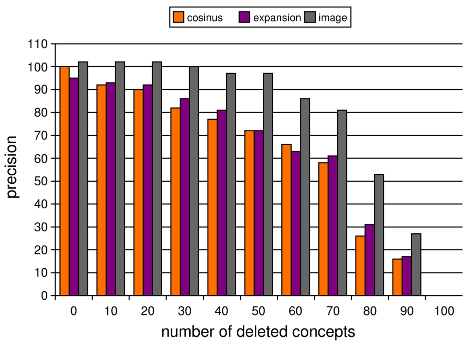
We compare our *image based method* with two others that are classicaly used in the context of a shared ontology. In the *cosine* based method, relevance is defined by the cosine between the query and the document vectors. In the *rough expansion* method, the effects of propagating weights from different concepts are mixed in a single vector. Relevance is obtained using the cosine. This method avoids some silence, but often generates too much noise, without any highly accurate sense disambiguation [10]. In the context of a shared ontology, our method shows *i)* better results than rough expansion and *ii)* results that are comparable with the cosine ones (a 2% increase of recall and precison). In the context of two different ontologies, the cosine based method is applied by the document provider p_2 in space $\mathcal{C}_{\Omega_2}$: in the query, only the shared concepts are considered. Rough expansion is done at the query intiator p_1's and the cosine is calculted at p_2's.

Because the manipulation of two different ontologies is a heavy process, we decided to *simulate* that use. Wordnet is used by both p_1 (to express its queries) and p_2 (to index its documents). However, in the result of the ontology mapping, we randomdly remove a given percentage of the shared concepts (from 10% to

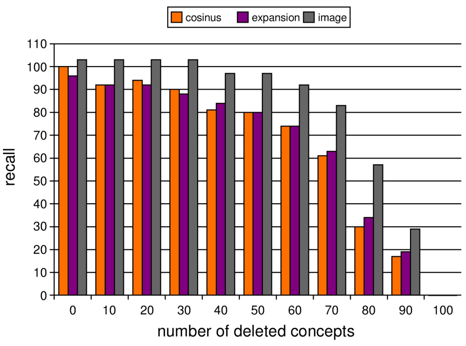
[1] It was collected between 1957 and 1968 by Cyril W. Cleverdon. The documents are abstracts of research papers.

90%). This amounts to simulate two peers that use the same ontology but are not aware of it. Of course, this eases interpretation. In particular, taking the lowest common ancestor of the shared concepts of $\overrightarrow{sed_c}$ to find the corresponding concept of central concept c gives good results most of the time.

Figure 5 shows the results obtained in average for the first twelve queries of the testing corpus. The reference method is the cosine one when no concept is removed, which gives a given reference precision and recall. Then, for each method and each percentage of removed concepts, we compute the ratio of the precision obtained (respectively recall) by the reference precision. When the percentage of randomly removed concepts increases, precision (figure 5 (a)) and recall (Figure 5 (b)) decrease i.e. the results are less and less relevant. However, our image and interpretation based solution shows much better results. When the percentage of removed concepts is under 70%, we still get 80% or more of the answers obtained in the reference case.



(a)



(b)

Fig. 5. Evolution of (a) precision and (b) recall in function of the percentage of concepts randomly removed from the set of shared concepts

6 Conclusion

The main contribution of this paper is a solution which improves information exchange between two peers that use different ontologies. Our solution uses semantic vectors to represent documents and queries. It only requires the peers to share some concepts and uses the unshared concepts to find additional relevant documents. To the best of our knowledge, the problem has never been addressed before and our approach is a first, encouraging solution.

When performing query expansion, the query initiator makes more precise the concepts of the query by associating an expansion to each of them (SED). The expansion depends on the initiator's characteristics: ontology, similarity, propagation function. Interpretation by the document provider is not easy because the peers do not share the vector space. Given its own ontology and similarity function, it first finds out a corresponding concept for the central concept of each SED, and then interprets the whole SED. The interpreted SEDs are used to compute an image of the documents and their relevance. This is only possible because the central concepts are expanded separately. The results of our preliminary experiments show that our approach significantly improves information exchange, finding up to 90% of the documents that would be found if all the concepts were shared.

As future work, we plan to conduct additional testing in different contexts to verify the robustness of our approach. Another aspect that we want to consider carefully is complexity to improve efficiency. Finally, we plan a theoretical study of the impact of interpretation when several peers are involved

References

1. Bidault, A., Froidevaux, C., Safar, B.: Repairing queries in a mediator approach. In: ECAI (2000)
2. Desmontils, E., Jacquin, C.: The Emerging Semantic Web. In: chapter Indexing a web site with a terminology oriented ontology (2002)
3. Euzenat, J., Shvaiko, P.: Ontology matching. Springer, Heidelberg (2007)
4. Gómez-Pérez, A., Fernández, M., Corcho, O.: Ontological Engineering. Springer, London (2004)
5. Ives, Z.G., Halevy, A.Y., Mork, P., Tatarinov, I.: Piazza: mediation and integration infrastructure for semantic web data. Journal of Web Semantics (2003)
6. Nie, J.-Y., Jin, F.: Integrating logical operators in query expansion invector space model. In: SIGIR workshop on Mathematical and Formal methods in Information Retrieval (2002)
7. Resnik, P.: Using information content to evaluate semantic similarity in a taxonomy. In: IJCAI (1995)
8. Rousset, M.-C.: Somewhere: a scalable p2p infrastructure for querying distributed ontologies. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 698–703. Springer, Heidelberg (2006)
9. Sanderson, M.: Retrieving with good sense. Information Retrieval (2000)
10. Voorhees, E.M.: Query expansion using lexical-semantic relations. In: SIGIR, Dublin (1994)
11. Woods, W.: Conceptual indexing: A better way to organize knowledge. Technical report, Sun Microsystems Laboratories (1997)

SPARQL++ for Mapping Between RDF Vocabularies*

Axel Polleres[1], François Scharffe[2], and Roman Schindlauer[3,4]

[1] DERI Galway, National University of Ireland, Galway
axel@polleres.net
[2] Leopold-Franzens Universität Innsbruck, Austria
francois.scharffe@uibk.ac.at
[3] Department of Mathematics, University of Calabria, 87030 Rende (CS), Italy
[4] Institut für Informationssysteme, Technische Universität Wien
roman@kr.tuwien.ac.at

Abstract. Lightweight ontologies in the form of RDF vocabularies such as SIOC, FOAF, vCard, etc. are increasingly being used and exported by "serious" applications recently. Such vocabularies, together with query languages like SPARQL also allow to syndicate resulting RDF data from arbitrary Web sources and open the path to finally bringing the Semantic Web to operation mode. Considering, however, that many of the promoted lightweight ontologies overlap, the lack of suitable standards to describe these overlaps in a declarative fashion becomes evident. In this paper we argue that one does not necessarily need to delve into the huge body of research on ontology mapping for a solution, but SPARQL itself might — with extensions such as external functions and aggregates — serve as a basis for declaratively describing ontology mappings. We provide the semantic foundations and a path towards implementation for such a mapping language by means of a translation to Datalog with external predicates.

1 Introduction

As RDF vocabularies like SIOC,[1] FOAF,[2] vCard,[3] etc. are increasingly being used and exported by "serious" applications we are getting closer to bringing the Semantic Web to operation mode. The standardization of languages like RDF, RDF Schema and OWL has set the path for such vocabularies to emerge, and the recent advent of an operable query language, SPARQL, gave a final kick for wider adoption. These ingredients allow not only to publish, but also to syndicate and reuse metadata from arbitrary distibuted Web resources in flexible, novel ways.

When we take a closer look at emerging vocabularies we realize that many of them overlap, but despite the long record of research on ontology mapping and alignment, a

* This research has been partially supported by the European Commission under the FP6 projects inContext (IST-034718), REWERSE (IST 506779), and Knowledge Web (FP6-507482), by the Austrian Science Fund (FWF) under project P17212-N04, as well as by Science Foundation Ireland under the Lion project (SFI/02/CE1/I131).

[1] http://sioc-project.org/

[2] http://xmlns.com/foaf/0.1/

[3] http://www.w3.org/TR/vcard-rdf

standard language for defining mapping rules between RDF vocabularies is still missing. As it turns out, the RDF query language SPARQL [24] itself is a promising candidate for filling this gap: Its CONSTRUCT queries may themselves be viewed as rules over RDF. The use of SPARQL as a rules language has several advantages: (i) the community is already familiar with SPARQL's syntax as a query language, (ii) SPARQL supports already a basic set of built-in predicates to filter results and (iii) SPARQL gives a very powerful tool, including even non-monotonic constructs such as OPTIONAL queries.

When proposing the use of SPARQL's CONSTRUCT statement as a rules language to define mappings, we should first have a look on existing proposals for syntaxes for rules languages on top of RDF(S) and OWL. For instance, we can observe that SPARQL may be viewed as syntactic extension of SWRL [16]: A SWRL rule is of the form $ant \Rightarrow cons$, where both antecedent and consequent are conjunctions of atoms $a_1 \wedge \ldots \wedge a_n$. When reading these conjunctions as basic graph patterns in SPARQL we might thus equally express such a rule by a CONSTRUCT statement:

$$\textsf{CONSTRUCT}\ \{\ cons\ \}\ \textsf{WHERE}\ \{\ ant\ \}$$

In a sense, such SPARQL "rules" are more general than SWRL, since they may be evaluated on top of arbitrary RDF data and — unlike SRWL — not only on top of valid OWL DL. Other rules language proposals, like WRL [7] or TRIPLE [8] which are based on F-Logic [18] Programming may likewise be viewed to be layerable on top of RDF, by applying recent results of De Bruijn et al. [5,6]. By the fact that (i) expressive features such as negation as failure which are present in some of these languages are also available in SPARQL [4] and (ii) F-Logic molecules in rule heads may be serialized in RDF again, we conjecture that rules in these languages can similarly be expressed as syntactic variants of SPARQL CONSTRUCT statements.[5]

On the downside, it is well-known that even a simple rules language such as SWRL already lead to termination/undecidability problems when mixed with ontology vocabulary in OWL without care. Moreover, it is not possible to express even simple mappings between common vocabularies such as FOAF[2] and VCard[3] in SPARQL only. To remedy this situation, we propose the following approach to enable complex mappings over ontologies: First, we keep the expressivity of the underlying ontology language low, restricting ourselves to RDFS, or, more strictly speaking to, ρdf^- ontologies (a variant of ρdf [20] defined in Subsection 4.5); second, we extend SPARQL's CONSTRUCT by features which are almost essential to express various mappings, namely: a set of useful built-in functions (such as string-concatenation and arithmetic functions on numeric literal values) and aggregate functions (min, max, avg). Third, we show that evaluating SPARQL queries on top of ρdf^- ontologies plus mapping rules is decidable by translating the problem to query answering over HEX-programs, i.e., logic programs with external built-ins using the answer-set semantics, which gives rise to implementations on top of existing rules engines such as dlvhex. A prototype of a SPARQL engine for evaluating queries over combined datasets consisting of ρdf^- and SPARQL mappings has been implemented and is avaiblable for testing online.[6]

[4] See [24, Section 11.4.1].
[5] With the exception of predicates with arbitrary arities.
[6] http://kr.tuwien.ac.at/research/dlvhex/

The remainder of this paper is structured as follows. We start with some motivating examples of mappings which can and can't be expressed with SPARQL CONSTRUCT queries in Section 2 and suggest syntactic extensions of SPARQL, which we call SPARQL++, in order to deal with the mappings that go beyond. In Section 3 we introduce HEX-programs, whereafter in Section 4 we show how SPARQL++ CONSTRUCT queries can be translated to HEX-programs, and thereby bridge the gap to implementations of SPARQL++. Next, we show how additional ontological inferences by ρdf$^-$ ontologies can be itself viewed as a set of SPARQL++ CONSTRUCT "mappings" to HEX-programs and thus embedded in our overall framework, evaluating mappings and ontological inferences at the same level, while retaining decidability. After a brief discussion of our current prototype and a discussion of related approaches, we conclude in Section 6 with an outlook to future work.

2 Motivating Examples – Introducing SPARQL

Most of the proposals in the literature for defining mappings between ontologies use subsumption axioms (by relating defining classes or (sub)properties) or bridge rules [3]. Such approaches do not go much beyond the expressivity of the underlying ontology language (mostly RDFS or OWL). Nonetheless, it turns out that these languages are insufficient for expressing mappings between even simple ontologies or when trying to map actual sets of data from one RDF vocabulary to another one. In Subsection 10.2.1 of the latest SPARQL specification [24] an example for such a mapping from FOAF to VCard is explicitly given, translating the VCard properties into the respective FOAF properties most of which could equally be expressed by simple rdfs:subPropertyOf statements. However, if we think the example a bit further, we quickly reach the limits of what is expressible by subclass- or subproperty statements.

Example 1. A simple and straightforward example for a mapping from `VCard:FN` to `foaf:name` is given by the following SPARQL query:

```
CONSTRUCT { ?X foaf:name ?FN . } WHERE { ?X VCard:FN ?FN . FILTER isLiteral(?FN) }
```

The filter expression here reduces the mapping by a kind of additional "type checking" where only those names are mapped which are merely given as a single literal.

Example 2. The situation becomes more tricky for other terms, for instance `VCard:n` (name) and `foaf:name`, because `VCard:n` consists of a substructure consisting of *Family name*, *Given name*, *Other names*, *honorific Prefixes*, and *honorific Suffixes*. One possibility is to concatenate all these to constitute a single `foaf:name`:

```
CONSTRUCT { ?X foaf:name ?Name . }
WHERE {?X VCard:N ?N .
        OPTIONAL {?N VCard:Family ?Fam } OPTIONAL {?N VCard:Given  ?Giv }
        OPTIONAL {?N VCard:Other  ?Oth } OPTIONAL {?N VCard:Prefix ?Prefix }
        OPTIONAL {?N VCard:Suffix ?Suffix }
        FILTER (?Name = fn:concat(?Prefix," ",?Giv, " ",?Fam," ",?Oth," ",?Suffix))
       }
```

We observe the following problem here: First, we use filters for constructing a new binding which is not covered by the current SPARQL specification, since filter expressions are not meant to create new bindings of variables (in this case the variable

?Name), but only filter existing bindings. Second, if we wanted to model the case where e.g., several other names were provided, we would need built-in functions beyond what SPARQL currently provides, in this case a string manipulation function such as fn:concat. SPARQL supplies a subset of the functions and operators defined by XPath/XQuery, but these cover only boolean functions, like arithmetic comparison operators and basic arithmetic functions but no string manipulation routines. Even with the full range of XPath/XQuery functions available, we would still have to slightly "extend" fn:concat here, assuming that unbound variables are handled properly, being replaced by an empty string in case one of the optional parts of the name structure is not defined.

Apart from built-in functions like string operations, aggregate functions such as count, minimum, maximum or sum, are another helpful construct for many mappings that is currently not available in SPARQL. Finally, although we can query and create new RDF graphs by SPARQL CONSTRUCT statements mapping one vocabulary to another, there is no well-defined way to combine such mappings with arbitrary data, especially when we assume that (1) mappings are not restricted to be unidirectional from one vocabulary to another, but bidirectional, and (2) additional ontological inferences such as subclass/subproperty relations defined in the mutually mapped vocabularies should be taken into account when querying over syndicated RDF data and mappings. Hence, we propose the following extensions of SPARQL:

- We introduce an extensible set of useful built-in and aggregate functions.
- We permit function calls and aggregates in the CONSTRUCT clause,
- We further allow CONSTRUCT queries nested in FROM statements, or more general, allowing CONSTRUCT queries as part of the dataset.

2.1 Built-In Functions and Aggregates in Result Forms

Considering Example 2, it would be more intuitive to carry out the string translation from VCard:n to foaf:name in the result form, i.e., in the CONSTRUCT clause:

```
CONSTRUCT {?X foaf:name fn:concat(?Prefix," ",?Giv," ",?Fam," ",?Oth," ",?Suffix).}
WHERE { ?X VCard:N ?N .
        OPTIONAL {?N VCard:Family ?Fam } OPTIONAL {?N VCard:Given  ?Giv }
        OPTIONAL {?N VCard:Other  ?Oth } OPTIONAL {?N VCard:Prefix ?Prefix }
        OPTIONAL {?N VCard:Suffix ?Suffix } }
```

Another example for a non-trivial mapping is the different treatment of telephone numbers in FOAF and VCard.

Example 3. A VCard:tel is a foaf:phone – more precisely, VCard:tel is related to foaf:phone as follows. VCard stores Telephone numbers as string literals, whereas FOAF uses resources, i.e., URIs with the tel: URI-scheme:

```
CONSTRUCT { ?X foaf:phone rdf:Resource(fn:concat("tel:",fn:encode-for-uri(?T)) . }
WHERE { ?X VCard:tel ?T . }
```

Here we assumed the availability of a cast-function, which converts an xs:string to an RDF resource. While the distinction between literals and URI references in RDF usually makes perfect sense, this example shows that conversions between URI references and literals become necessary by practical uses of RDF vocabularies.

The next example shall illustrate the need for aggregate functions in mappings.

Example 4. The Description of a Project (DOAP) vocabulary[7] contains revision, i.e. version numbers of released versions of projects. With an aggregate function `MAX`, one can map DOAP information into the RDF Open Source Software Vocabulary [8], which talks about the latest release of a project, by picking the maximum value (numerically or lexicographically) of the set of revision numbers specified by a graph pattern as follows:

```
CONSTRUCT {?P os:latestRelease MAX(?V : ?P doap:release ?R. ?R doap:revision ?V)}
WHERE {?P rdf:type doap:Project. }
```

Here, the WHERE clause singles out all projects, while the aggregate selects the highest (i.e., latest) revision date of any available version for that project.

2.2 Nested CONSTRUCT Queries in FROM Clauses

The last example shows another example of "aggregation" which is not possible with SPARQL upfront, but may be realized by nesting CONSTRUCT queries in the FROM clause of a SPARQL query.

Example 5. Imagine you want to map/infer from an ontology having co-author relationships declared using `dc:creator` properties from the Dublin Core metadata vocabulary to `foaf:knows`, i.e., you want to specify "*If $?a$ and $?b$ have co-authored the same paper, then $?a$ knows $?b$*". The problem here is that a mapping using CONSTRUCT clauses needs to introduce new blank nodes for both $?a$ and $?b$ (since `dc:creator` is a datatype property usually just giving the name string of the author) and then need to infer the knows relation, so what we really want to express is a mapping

> If $?a$ and $?b$ are `dc:creators` of the same paper, then someone named with `foaf:name` $?a$ `foaf:knows` someone with `foaf:name` $?b$.

A first-shot solution could be:

```
CONSTRUCT { _:a foaf:knows _:b . _:a foaf:name ?n1 . _:b foaf:name ?n2 . }
FROM <g>  WHERE { ?p dc:creator ?n1 . ?p dc:creator ?n2 . FILTER ( ?n1 != ?n2 ) }
```

Let us consider the present paper as example graph `g`:

```
g: <http://ex.org/papers#sparqlmappingpaper> dc:creator "Axel"
   <http://ex.org/papers#sparqlmappingpaper> dc:creator "Roman"
   <http://ex.org/papers#sparqlmappingpaper> dc:creator "Francois"
```

By the semantics of blank nodes in CONSTRUCT clauses — SPARQL creates new blank node identifiers for each solutions set matching the WHERE clause — the above would infer the following additional triples:

```
 _:a1 foaf:knows _:b1.  _:a1 foaf:name  "Axel".     _:b1 foaf:name  "Roman".
 _:a2 foaf:knows _:b2.  _:a2 foaf:name  "Axel".     _:b2 foaf:name  "Francois".
 _:a3 foaf:knows _:b3.  _:a3 foaf:name  "Francois". _:b3 foaf:name  "Roman".
 _:a4 foaf:knows _:b4.  _:a4 foaf:name  "Francois". _:b4 foaf:name  "Axel".
 _:a5 foaf:knows _:b5.  _:a5 foaf:name  "Roman".    _:b5 foaf:name  "Axel".
 _:a6 foaf:knows _:b6.  _:a6 foaf:name  "Roman".    _:b6 foaf:name  "Francois".
```

[7] http://usefulinc.com/doap/

[8] http://xam.de/ns/os/

Obviously, we lost some information in this mapping, namely the corellations that the "Axel" knowing "Francois" is the same "Axel" that knows "Roman", etc. We could remedy this situation by allowing to nest CONSTRUCT queries in the FROM clause of SPARQL queries as follows:

```
CONSTRUCT { ?a knows ?b . ?a foaf:name ?aname . ?b foaf:name ?bname . }
FROM { CONSTRUCT { _:auth foaf:name ?n . ?p aux:hasAuthor _:auth . }
       FROM <g> WHERE { ?p dc:creator ?n . } }
WHERE { ?p aux:hasAuthor ?a . ?a foaf:name ?aname .
        ?p aux:hasAuthor ?b . ?b foaf:name ?bname . FILTER ( ?a != ?b ) }
```

Here, the "inner" CONSTRUCT creates a graph with unique blank nodes for each author per paper, whereas the outer CONSTRUCT aggregates a more appropriate answer graph:

```
_:auth1 foaf:name "Axel". _:auth2 foaf:name "Roman". _:auth3 foaf:name "Francois".
_:auth1 foaf:knows _:auth2. _:auth1 foaf:knows _:auth3.
_:auth2 foaf:knows _:auth1. _:auth2 foaf:knows _:auth3.
_:auth3 foaf:knows _:auth1. _:auth3 foaf:knows _:auth2.
```

In Section 4, we will extend SPARQL to deal with these features. This extended version of the language, which we call SPARQL++ shall allow to evaluate SPARQL queries on top of RDF(S) data combined with mappings again expressed in SPARQL++.

3 Preliminaries – HEX-Programs

To evaluate SPARQL++ queries, we will translate them to so-called HEX-*programs* [27], an extension of logic programs under the answer-set semantics.

Let $Pred$, $Const$, Var, $exPr$ be mutually disjoint sets of predicate, constant, variable symbols, and external predicate names, respectively. In accordance with common notation in LP and the notation for external predicates from [9] we will in the following assume that $Const$ comprises the set of numeric constants, string constants beginning with a lower case letter, or double-quoted string literals, and IRIs.[9] Var is the set of string constants beginning with an uppercase letter. Elements from $Const \cup Var$ are called *terms*. Given $p \in Pred$ an *atom* is defined as $p(t_1, \ldots, t_n)$, where n is called the arity of p and $t_1, \ldots, t_n$ are terms. An *external atom* is of the form

$$g[Y_1, \ldots, Y_n](X_1, \ldots, X_m),$$

where $Y_1, \ldots, Y_n$ is a list of predicates and terms and $X_1, \ldots, X_m$ is a list of terms (called *input list* and *output list*, respectively), and $g \in exPr$ is an external predicate name. We assume the *input* and *output arities* n and m fixed for g. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates and terms. Note that this means that external predicates, unlike usual definitions of built-ins in logic programming, can not only take constant parameters but also (extensions of) predicates as input.

Definition 1. *A* rule *is of the form*

$$h \leftarrow b_1, \ldots, b_m, not\, b_{m+1}, \ldots not\, b_n \tag{1}$$

[9] For the purpose of this paper, we will disregard language-tagged and datatyped literals in the translation to HEX-programs.

where h and b_i ($m+1 \leq i \leq n$) are atoms, b_k ($1 \leq k \leq m$) are either atoms or external atoms, and 'not' is the symbol for negation as failure.

We use $H(r)$ to denote the head atom h and $B(r)$ to denote the set of all body literals $B^+(r) \cup B^-(r)$ of r, where $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$.

The notion of input and output terms in external atoms described above denotes the binding pattern. More precisely, we assume the following condition which extends the standard notion of safety (cf. [28]) in Datalog with negation.

Definition 2 (Safety). *Each variable appearing in a rule must appear in a non-negated body atom or as an output term of an external atom.*

Finally, we define HEX-programs.

Definition 3. *A* HEX-program *P is defined as a set of safe rules r of the form* (1).

The notions of *grounding*, *Herbrand Base* and *interpretation* correspond to traditional logic programming. With every external predicate name $e \in exPr$ we associate an $(n{+}m{+}1)$-ary Boolean function f_e assigning each tuple $(I, y_1, \ldots, y_n, x_1, \ldots, x_m)$ either 0 or 1, where n/m are the input/output arities of e, $I \subseteq HB_P$, $x_i \in Const$, and $y_j \in Pred \cup Const$. We say that $I \subseteq HB_P$ is a *model* of a ground external atom $a = e[y_1, \ldots, y_n](x_1, \ldots, x_m)$, denoted $I \models a$, iff $f_e(I, y_1 \ldots, y_n, x_1, \ldots, x_m) = 1$.

Let r be a ground rule. We define (i) $I \models H(r)$ iff there is some $a \in H(r)$ such that $I \models a$, (ii) $I \models B(r)$ iff $I \models a$ for all $a \in B^+(r)$ and $I \not\models a$ for all $a \in B^-(r)$, and (iii) $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$. We say that I is a *model* of a HEX-program P, denoted $I \models P$, iff $I \models r$ for all $r \in ground(P)$.

The semantics we use here generalizes the answer-set semantics [13] and is defined using the *FLP-reduct* [12], which—contrary to the traditional Gelfond-Lifschitz reduct—ensures minimality of answer sets also in presence of external atoms: The *FLP-reduct* of P with respect to $I \subseteq HB_P$, denoted P^I, is the set of all $r \in ground(P)$ such that $I \models B(r)$. $I \subseteq HB_P$ is an *answer set of* P iff I is a minimal model of P^I.

By the *cautious extension* of a predicate p we denote the set of atoms with predicate symbol p in the intersection of all answer sets of P.

For our purposes, we define a set of external predicates $exPr = \{rdf, isBLANK, isIRI, isLITERAL, =, !=, REGEX, CONCAT, COUNT, MAX, MIN, SK\}$ with a fixed semantics as follows. These external predicates exemplarily demonstrate that HEX-programs are expressive enough to model all the necessary ingredients for evaluating SPARQL filters ($isBLANK, isIRI, isLITERAL, =, !=, REGEX$) and also for more expressive built-in functions and aggregates ($CONCAT, SK, COUNT, MAX, MIN$). Here, $CONCAT$ is just an example built-in, assuming that more XPath/XQuery functions could similarly be added.

For the rdf predicate we write atoms as $rdf[i](s,p,o)$ with an input term $i \in Const \cup Var$ and output terms $s, p, o \in Const$. The external atom $rdf[i](s,p,o)$ is true if (s,p,o) is an RDF triple *entailed* by the RDF graph at IRI i. For the moment, here we consider simple RDF entailment [15] only.

The atoms $isBLANK[c](val)$, $isIRI[c](val)$, $isLITERAL[c](val)$ test input term $c \in Const \cup Var$ for being a valid string representation of a blank node, IRI reference or RDF literal. The atom $REGEX[c_1, c_2](val)$ test whether c_1 matches the regular expression c_2. All these external predicates return an output value $val \in \{\mathtt{t}, \mathtt{f}, \mathtt{e}\}$, representing truth, falsity or an error, following the semantics defined in [24, Sec. 11.3].

Apart from these truth-valued external atoms we add other external predicates which mimic built-in functions an aggregates. As an example predicate for a built-in, we chose the predicate $CONCAT[c_1, \ldots, c_n](c_{n+1})$ with variable input arity which concatenates string constants $c_1, \ldots, c_n$ into c_{n+1} and thus implements the semantics of `fn:concat` in XPath/XQuery [19].

Next, we define external predicates which mimic aggregate functions over a certain predicate. Let $p \in Pred$ with arity n, and $x_1, \ldots, x_n \in Const \cup \{mask\}$ where $mask$ is a special constant not allowed to appear anywhere except in input lists of aggregate predicates. Then $COUNT[p, x_1, \ldots, x_n](c)$ is true if c equals the number of distinct tuples $(t_1, \ldots, t_n)$, such that $I \models p(t_1, \ldots, t_n)$ and for all x_i different from the constant $mask$ it holds that $t_i = x_i$. $MAX[p, x_1, \ldots, x_n](c)$ (and $MIN[p, x_1, \ldots, x_n](c)$, resp.) is true if among all tuples $(t_1, \ldots, t_n)$, such that $I \models p(t_1, \ldots, t_n)$, c is the lexicographically greatest (smallest, resp.) value among all the t_i such that $x_i = mask$.[10]

We will illustrate the use of these external predicates to express aggregations below when discussing the actual translation from SPARQL++ to HEX-programs.

Finally, the external predicate $SK[id, v_1, \ldots, v_n](sk_{n+1})$ computes a unique, new "Skolem"-like term $id(v_1, \ldots, v_n)$ from its input parameters. We will use this built-in function in our translation of SPARQL queries with blank nodes in the CONSTRUCT part. Similar to the aggregate functions mentioned before, when using SK we will need to take special care in our translation in order to retain strong safety.

As widely known, for programs without external predicates, safety guarantees that the number of entailed ground atoms is finite. Though, by external atoms in rule bodies, new, possibly infinitly many, ground atoms could be generated, even if all atoms themselves are safe. In order to avoid this, a stronger notion of safety for HEX-programs is defined in [27]: Informally, this notion says that a HEX-program is *strongly safe*, if no external predicate recursively depends on itself, thus defining a notion of stratification over external predicates. Strong safety guarantees finiteness of models as well as finite computability of external atoms.

4 Extending SPARQL Towards Mappings

In Section 2 we have shown that an extension of the CONSTRUCT clause is needed for SPARQL to be suitable for mapping tasks. In the following, we will formally define extended SPARQL queries which allow to integrate built-in functions and aggregates in CONSTRUCT clauses as well as in FILTER expressions. We will define the semantics of such extended queries, and, moreover, we will provide a translation to HEX-programs, building upon an existing translation presented in [22].

A SPARQL++ *query* $Q = (R, P, DS)$ consists of a result form R, a graph pattern P, and an extended dataset DS as defined below.[11] We refer to [24] for syntactical details and will explain these in the following as far as necessary.

[10] Note that in this definition we allow min/max to aggregate over several variables.

[11] As we deal mainly with CONSTRUCT queries here, we will ignore solution modifiers.

For a SELECT query, a result form R is simply a set of variables, whereas for a CONSTRUCT query, the result form R is a set of triple patterns.

We assume the pairwise disjoint, infinite sets I, B, L and Var, which denote IRIs, blank node identifiers, RDF literals, and variables respectively. $I \cup L \cup Var$ is also called the set of *basic RDF terms*. In this paper, we allow as blank node identifiers nested ground terms similar to HiLog terms [4], such that B is defined recursively over an infinite set of constant blank node identifiers B_c as follows:

– each element of B_c is a blank node identifier, i.e., $B_c \subseteq B$.
– for $b \in B$ and $t_1, \ldots, t_n$ in $I \cup B \cup L$, $b(t_1, \ldots, t_n) \in B$.

Now, we extend the SPARQL syntax by allowing built-in functions and aggregates in place of basic RDF terms in graph patterns (and thus also in CONSTRUCT clauses) as well as in filter expressions. We define the set Blt of *built-in terms* as follows:

– All basic terms are built-in terms.
– If blt is a built-in predicate (e.g., `fn:concat` from above or another XPath/ XQuery functions), and $c_1, \ldots, c_n$ are built-in terms then $blt(c_1, \ldots, c_n)$ is a built-in term.
– If agg is an aggregate function (e.g., $COUNT$, MIN, MAX), P a graph pattern, and V a tuple of variables appearing in P, then $agg\,\langle V : P\rangle$ is a built-in term.[12]

In the following we will introduce extended graph patterns that may include built-in terms and extended datasets that can be constituted by CONSTRUCT queries.

4.1 Extended Graph Patterns

As for *graph patterns*, we follow the recursive definition from [21]:

– A triple pattern (s, p, o) is a graph pattern where $s, o \in Blt$ and $p \in I \cup Var$.[13] Triple patterns which only contain basic terms are called *basic triple patterns* and *value-generating triple patterns* otherwise.
– Let P_1, P_2 be graph patterns, $i \in I \cup Var$, R a filter expression, then $(P_1 \text{ FILTER } R)$, $(P_1 \text{ OPT } P_2)$, $(P_1 \text{ UNION } P_2)$, $(\text{GRAPH } i\ P_1)$, and $(P_1 \text{ AND } P_2)$ are graph patterns.[14]

For any pattern P, we denote by $vars(P)$ the set of all variables occurring in P and by $\overline{vars}(P)$ the tuple obtained by the lexicographic ordering of all variables in P. As *atomic filter expression*, we allow here the unary predicates BOUND (possibly with variables as arguments), isBLANK, isIRI, isLITERAL, and binary equality predicates '$=$' with arbitrary safe built-in terms as arguments. *Complex filter expressions* can be built using the connectives '$\neg$', '$\wedge$', and '$\vee$'.

Similar to aggregates in logic programming, we use a notion of safety. First, given a query $Q = (R, P, DS)$ we allow only basic triple patterns in P, ie. we only allow built-ins and aggregates only in FILTERs or in the result pattern R. Second, a built-in term blt occurring in the result form or in P in a query $Q = (R, P, DS)$ is *safe* if all variables recursively appearing in blt also appear in a basic triple pattern within P .

[12] This aggregate syntax is adapted from the resp. definition for aggregates in LP from [12].
[13] We do not consider blanks nodes here as these can be equivalently replaced by variables [5].
[14] We use AND to keep with the operator style of [21] although it is not explicit in SPARQL.

4.2 Extended Datasets

In order to allow the definition of RDF data side-by-side with implicit data defined by mappings of different vocabularies or, more general, views within RDF, we define an *extended RDF graph* as a set of RDF triples $I \cup L \cup B \times I \times I \cup L \cup B$ and CONSTRUCT queries. An RDF graph (or dataset, resp.) without CONSTRUCT queries is called a *basic graph* (or dataset, resp.).

The dataset $DS = (G, \{(g_1, G_1), \dots (g_k, G_k)\})$ of a SPARQL query is defined by (i) a default graph G, i.e., the RDF merge [15, Section 0.3] of a set of extended RDF graphs, plus (ii) a set of named graphs, i.e., pairs of IRIs and corresponding extended graphs. Without loss of generality (there are other ways to define the dataset such as in a SPARQL protocol query), we assume DS defined by the IRIs given in a set of FROM and FROM NAMED clauses. As an exception, we assume that any CONSTRUCT query which is part of an extended graph G by default (i.e., in the absence of FROM and FROM NAMED clauses) has the dataset $DS = (G, \emptyset)$ For convenience, we allow extended graphs consisting of a single CONSTRUCT statement to be written directly in the FROM clause of a SPARQL++ query, like in Example 5.

We will now define syntactic restrictions on the CONSTRUCT queries allowed in extended datasets, which retain finite termination on queries over such datasets. Let G be an extended graph. First, for any CONSTRUCT query $Q = (R, P, DS_Q)$ in G, DS_Q we allow only triple patterns $tr = (s, p, o)$ in P or R where $p \in I$, i.e., neither blank nodes nor variables are allowed in predicate positions in extended graphs, and, additionally, $o \in I$ for all triples such that $p =$ `rdf:type`. Second, we define a predicate-class-dependency graph over an extended dataset $DS = (G, \{(g_1, G_1), \dots (g_k, G_k)\})$ as follows. The predicate-class-dependency graph for DS has an edge $p \rightarrow r$ with $p, r \in I$ for any CONSTRUCT query $Q = (R, P, DS)$ in G with r (or p, resp.) either (i) a predicate different from `rdf:type` in a triple in R (or P, resp.), or (ii) an object in an `rdf:type` triple in R (or P, resp.). All edges such that r occurs in a value-generating triple are marked with '$*$'. We now say that DS is *strongly safe* if its predicate-class-dependency graph does not contain any cycles involving marked edges. As it turns out, in our translation in Section 4.4 below, this condition is sufficient (but not necessary) to guarantee that any query can be translated to a strongly safe HEX-program.

Like in [26] we assume that blank node identifiers in each query $Q = (R, P, DS)$ have been standardized apart, i.e., that no blank nodes with the same identifiers appear in a different scope. The scope of a blank node identifier is defined as the graph or graph pattern it appears in, where each WHERE or CONSTRUCT clause open a "fresh" scope . For instance, take the extended graph dataset in Fig. 1(a), its standardized apart version is shown in Fig. 1(b). Obviously, extended datasets can always be standardized apart in linear time in a preprocessing step.

4.3 Semantics

The semantics of SPARQL++ is based on the formal semantics for SPARQL by Pérez et al. in [21] and its translation into HEX-programs in [22]. We denote by T_{null} the union

```
g1:  :paper2 foaf:maker _:a.
     _:a foaf:name "Jean Deau".

g2: :paper1 dc:creator "John Doe".
    :paper1 dc:creator "Joan Dough".
    CONSTRUCT {_:a foaf:knows _:b .
               _:a foaf:name ?N1 .
               _:b foaf:name ?N2 . }
    WHERE {?X dc:creator ?N1,?N2.
           FILTER( ?N1 != ?N2 ) }
```

(a)

```
g1:  :paper2 foaf:maker _:b1.
     _:b1 foaf:name "Jean Deau".

g2: :paper1 dc:creator "John Doe".
    :paper1 dc:creator "Joan Dough".
    CONSTRUCT {_:b2 foaf:knows _:b3 .
               _:b2 foaf:name ?N1 .
               _:b3 foaf:name ?N2 . }
    WHERE {?X dc:creator ?N1,?N2.
           FILTER( ?N1 != ?N2 ) }
```

(b)

Fig. 1. Standardizing apart blank node identifiers in extended datasets

$I \cup B \cup L \cup \{\mathsf{null}\}$, where null is a dedicated constant denoting the unknown value not appearing in any of I, B, or L, how it is commonly introduced when defining outer joins in relational database systems. A *substitution* θ from Var to T_{null} is a partial function $\theta : Var \rightarrow T_{\mathsf{null}}$. We write substitutions in postfix notation in square brackets, i.e., if $t, t' \in Blt$ and $v \in Var$, then $t[v/t']$ is the term obtained from replacing all occcurences of v in t by t'. The *domain* of θ, $dom(\theta)$, is the subset of Var where θ is defined. The lexicographic ordering of this subset is denoted by $\overline{dom}(Var)$. For a substitution θ and a set of variables $D \subseteq Var$ we define the substitution θ^D with domain D as follows

$$x\theta^D = \begin{cases} x\theta & \text{if } x \in dom(\theta) \cap D \\ \mathsf{null} & \text{if } x \in D \setminus dom(\theta) \end{cases}$$

Let $x \in Var$, θ_1, θ_2 be substitutions, then $\theta_1 \cup \theta_2$ is the substitution obtained as follows:

$$x(\theta_1 \cup \theta_2) = \begin{cases} & x\theta_1 & \text{if } x\theta_1 \text{ defined and } x\theta_2 \text{ undefined} \\ \text{else:} & x\theta_1 & \text{if } x\theta_1 \text{ defined and } x\theta_2 = \mathsf{null} \\ \text{else:} & x\theta_2 & \text{if } x\theta_2 \text{ defined} \\ \text{else:} & \text{undefined} & \end{cases}$$

Thus, in the union of two substitutions defined values in one take precedence over null values the other substitution. Two substitutions θ_1 and θ_2 are *compatible* when for all $x \in dom(\theta_1) \cap dom(\theta_2)$ either $x\theta_1 = \mathsf{null}$ or $x\theta_2 = \mathsf{null}$ or $x\theta_1 = x\theta_2$ holds, i.e., when $\theta_1 \cup \theta_2$ is a substitution over $dom(\theta_1) \cup dom(\theta_2)$. Analogously to Pérez et al. we define join, union, difference, and outer join between two sets of substitutions Ω_1 and Ω_2 over domains D_1 and D_2, respectively:

$$\begin{aligned} \Omega_1 \bowtie \Omega_2 &= \{\theta_1 \cup \theta_2 \mid \theta_1 \in \Omega_1, \theta_2 \in \Omega_2, \theta_1 \text{ and } \theta_2 \text{ are compatible}\} \\ \Omega_1 \cup \Omega_2 &= \{\theta \mid \exists \theta_1 \in \Omega_1 \text{ with } \theta = \theta_1^{D_1 \cup D_2} \text{ or } \exists \theta_2 \in \Omega_2 \text{ with } \theta = \theta_2^{D_1 \cup D_2}\} \\ \Omega_1 - \Omega_2 &= \{\theta \in \Omega_1 \mid \text{ for all } \theta_2 \in \Omega_2,\ \theta \text{ and } \theta_2 \text{ not compatible}\} \\ \Omega_1 ⟕ \Omega_2 &= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 - \Omega_2) \end{aligned}$$

Next, we define the application of substitutions to built-in terms and triples: For a built-in term t, by $t\theta$ we denote the value obtained by applying the substitution to all variables in t. By $eval_\theta(t)$ we denote the value obtained by (i) recursively evaluating all built-in and aggregate functions, and (ii) replacing all bNode identifiers by complex bNode identifiers according to θ, as follows:

$eval_\theta(\texttt{fn:concat}(c_1, c_2, \ldots, c_n))$	Returns the `xs:string` that is the concatenation of the values of $c_1\theta,\ldots,c_1\theta$ after conversion. If any of the arguments is the empty sequence or null, the argument is treated as the zero-length string.
$eval_\theta(\texttt{COUNT}(V : P))$	Returns the number of distinct[15] answer substitutions for the query $Q = (V, P\theta, DS)$ where DS is the dataset of the encapsulating query.
$eval_\theta(\texttt{MAX}(V : P))$	Returns the maximum (numerically or lexicographically) of distinct answer substitutions for the query $Q = (V, P\theta, DS)$.
$eval_\theta(\texttt{MIN}(V : P))$	Analogous to `MAX`, but returns the minimum.
$eval_\theta(t)$	Returns $t\theta$ for all $t \in I \cup L \cup Var$, and $t(\overline{dom}(\theta)\theta)$ for $t \in B$.[16]

Finally, for a triple pattern $tr = (s, p, o)$ we denote by $tr\theta$ the triple $(s\theta, p\theta, o\theta)$, and by $eval_\theta(tr)$ the triple $(eval_\theta(s), eval_\theta(p), eval_\theta(o))$.

The evaluation of a graph pattern P over a basic dataset $DS = (G, G_n)$, can now be defined recursively by sets of substitutions, extending the definitions in [21,22].

Definition 4. *Let $tr = (s, p, o)$ be a basic triple pattern, P, P_1, P_2 graph patterns, and $DS = (G, G_n)$ a basic dataset, then the* evaluation $[[\cdot]]_{DS}$ *is defined as follows:*

$$
\begin{aligned}
[[tr]]_{DS} &= \{\theta \mid dom(\theta) = vars(P) \text{ and } tr\theta \in G\}\\
[[P_1 \textsf{ AND } P_2]]_{DS} &= [[P_1]]_{DS} \bowtie [[P_2]]_{DS}\\
[[P_1 \textsf{ UNION } P_2]]_{DS} &= [[P_1]]_{DS} \cup [[P_2]]_{DS}\\
[[P_1 \textsf{ OPT } P_2]]_{DS} &= [[P_1]]_{DS} ⟕ [[P_2]]_{DS}\\
[[\textsf{GRAPH } i\ P]]_{DS} &= [[P]]_{(i,\emptyset)}, \text{ for } i \in G_n\\
[[\textsf{GRAPH } v\ P]]_{DS} &= \{\theta \cup [v/g] \mid g \in G_n \text{ and } \theta \in [[P[v/g]\,]]_{(g,\emptyset)}\}, \text{ for } v \in Var\\
[[P \textsf{ FILTER } R]]_{DS} &= \{\theta \in [[P]]_{DS} \mid R\theta = \top\}
\end{aligned}
$$

Let R be a filter expression, $u, v \in Blt$. The valuation of R on a substitution θ, written $R\theta$ takes one of the three values $\{\top, \bot, \varepsilon\}$[17] and is defined as follows.

$R\theta = \top$, *if:* (1) $R = \textsf{BOUND}(v)$ *with* $v \in dom(\theta) \ \wedge\ eval_\theta(v) \neq \textsf{null}$;
(2) $R = \textsf{isBLANK}(v)$ *with* $eval_\theta(v) \in B$;
(3) $R = \textsf{isIRI}(v)$ *with* $eval_\theta(v) \in I$;
(4) $R = \textsf{isLITERAL}(v)$ *with* $eval_\theta(v) \in L$;
(5) $R = (u = v)$ *with* $eval_\theta(u) = eval_\theta(v) \wedge\ eval_\theta(u) \neq \textsf{null}$;
(6) $R = (\neg R_1)$ *with* $R_1\theta = \bot$;
(7) $R = (R1 \vee R2)$ *with* $R_1\theta = \top \ \vee\ R_2\theta = \top$;
(8) $R = (R1 \wedge R2)$ *with* $R_1\theta = \top \ \wedge\ R_2\theta = \top$.

$R\theta = \varepsilon$, *if:* (1) $R = \textsf{isBLANK}(v), R = \textsf{isIRI}(v), R = \textsf{isLITERAL}(v)$, *or*
$R = (u = v)$ *with* $(v \in Var \wedge v \notin dom(\theta)) \ \vee\ eval_\theta(v) = \textsf{null} \ \vee$
$(u \in Var \wedge u \notin dom(\theta)) \ \vee\ eval_\theta(u) = \textsf{null}$;
(2) $R = (\neg R_1)$ *and* $R_1\theta = \varepsilon$;
(3) $R = (R_1 \vee R_2)$ *and* $R_1\theta \neq \top \ \wedge\ R_2\theta \neq \top \ \wedge\ (R_1\theta = \varepsilon \ \vee\ R_2\theta = \varepsilon)$;
(4) $R = (R1 \wedge R2)$ *and* $R_1\theta = \varepsilon \ \vee\ R_2\theta = \varepsilon$.

$R\theta = \bot$ *otherwise.*

In [22] we have shown that the semantics defined this way corresponds with the original semantics for SPARQL defined in [21] without complex built-in and aggregate terms and on basic datasets.[18]

[15] Note that we give a set based semantics to the counting built-in, we do not take into account duplicate solutions which can arise from the multi-set semantics in [24] when counting.

[16] For blank nodes $eval_\theta$ constructs a new blank node identifier, similar to Skolemization.

[18] Our definition here only differs in in the application of $eval_\theta$ on built-in terms in filter expressions which does not make a difference if only basic terms appear in FILTERs.

Note that, so far we have only defined the semantics in terms of a pattern P and basic dataset DS, but neither taken the result form R nor extended datasets into account. As for the former, we proceed with formally define solutions for SELECT and CONSTRUCT queries, respectively. The semantics of a SELECT query $Q = (V, P, DS)$ is fully determined by its *solution tuples* [22].

Definition 5. *Let* $Q = (R, P, DS)$ *be a* SPARQL++ *query, and* θ *a substitution in* $[\![P]\!]_{DS}$*, then we call the tuple* $\overline{vars(P)}\theta$ *a solution tuple of* Q.

As for a CONSTRUCT queries, we define the *solution graphs* as follows.

Definition 6. *Let* $Q = (R, P, DS)$ *be a* SPARQL CONSTRUCT *query where blank node identifiers in* DS *and* R *have been standardized apart and* $R = \{t_1, \ldots, t_n\}$ *is the result graph pattern. Further, for any* $\theta \in [\![P]\!]_{DS}$*, let* $\theta' = \theta^{vars(R) \cup vars(P)}$*. The solution graph for* Q *is then defined as the triples obtained from*

$$\bigcup_{\theta in [\![P]\!]_{DS}} \{eval_{\theta'}(t_1), \ldots, eval_{\theta'}(t_n)\}$$

by eliminating all non-valid RDF triples.[19]

Our definitions so far only cover basic datasets. Extended datasets, which are implicitly defined bring the following additional challenges: (i) it is not clear upfront which blank node identifiers to give to blank nodes resulting from evaluating CONSTRUCT clauses, and (ii) extended datasets might involve recursive CONSTRUCT definitions which construct new triples in terms of the same graph in which they are defined. As for (i), we remedy the situation by constructing new identifier names via a kind of Skolemization, as defined in the function $eval_\theta$, see the table on page 889. $eval_\theta$ generates a unique blank node identifier for each solution θ. Regarding (ii) we avoid possibly infinite datasets over recursive CONSTRUCT clauses by the strong safety restriction in Section 4.2. Thus, we can define a translation from extended datasets to HEX-programs which uniquely identifies the solutions for queries over extended datasets.

4.4 Translation to HEX-Programs

Our translation from SPARQL++ queries to HEX-programs is based on the translation for non-extended SPARQL queries outlined in [22]. Similar to the well-known correspondence between SQL and Datalog, SPARQL++ queries can be expressed by HEX-programs, which provide the additional machinery necessary for importing and processing RDF data as well as evaluating built-ins and aggregates. The translation consists of two basic parts: (i) rules that represent the query's graph pattern (ii) rules defining the triples in the extended datasets.

We have shown in [22] that solution tuples for any query Q can be generated by a logic program and are represented by the extension of a designated predicate $\texttt{answer}_Q$, assuming that the triples of the dataset are available in a predicate $\texttt{triple}_Q$. We refer to [22] for details and only outline the translation here by examples.

[19] That is, triples with null values or blank nodes in predicate position, etc.

Complex graph patterns can be translated recursively in a rather straightforward way, where unions and join of graph patterns can directly be expressed by appropriate rule constructions, whereas OPTIONAL patterns involve negation as failure.

Example 6. Let query q select all persons who do not know anybody called "John Doe" from the extended dataset $DS = (g1 \cup g2, \emptyset)$, i.e., the merge of the graphs in Fig. 1(b).

```
SELECT  ?P FROM <g1> FROM <g2>
WHERE { ?P rdf:type foaf:Agent . FILTER ( !BOUND(?P1) )
        OPTIONAL { P? foaf:knows ?P1 . ?P1 foaf:name "John Doe" . } }
```

This query can be translated to the following HEX-program:

```
answer_q(P) :- answer1_q(P,P1), P1 = null.
answer1_q(P,P1) :- answer2_q(P), answer_q3(P,P1).
answer1_q(P,null) :- answer2_q(P), not answer3_q'(P).
answer2_q(P) :- triple_q(P,rdf:type,foaf:Agent,def).
answer3_q(P,P1) :- triple_q(P,foaf:knows,P1,def),triple(P1,foaf:name,"John Doe",def).
answer3_q'(P) :- answer3_q(P,P1).
```

More complex queries with nested patterns can be translated likewise by introducing more auxiliary predicates. The program part defining the $\texttt{triple}_q$ predicate fixes the triples of the dataset, by importing all explicit triples in the dataset as well as recursively translating all CONSTRUCT clauses and subqueries in the extended dataset.

Example 7. The program to generate the dataset triples for the extended dataset $DS = (g1 \cup g2, \emptyset)$ looks as follows:

```
triple_q(S,P,O,def) :- rdf["g1"](S,P,O).
triple_q(S,P,O,def) :- rdf["g2"](S,P,O).
triple_q(B2,foaf:knows,B3,def) :- SK[b2(X,N1,N2)](B2),SK[b3(X,N1,N2)](B3),
                                  answer_{C_1,g2}(X,N1,N2).
triple_q(B2,foaf:name,N1,def) :- SK[b2(X,N1,N2)](B2), answer_{C_1,g2}(X,N1,N2).
triple_q(B3,foaf:knows,N2,def) :- SK[b3(X,N1,N2)](B3), answer_{C_1,g2}(X,N1,N2).
answer_{C_1,g2}(X,N1,N2) :- triple_q(X,dc:creator, N1,def),
                           tripleq(X,dc:creator,N2,def), N1 != N2.
```

The first two rules import all triples given explicitly in graphs $g1, g2$ by means of the "standard" RDF import HEX predicate. The next three rules create the triples from the CONSTRUCT in graph $g2$, where the query pattern is translated by an own subprogram defining the predicate $\texttt{answer}_{C_1,g2}$, which in this case only consists of a single rule.

The example shows the use of the external function SK to create blank node ids for each solution tuple as mentioned before, which we need to emulate the semantics of blank nodes in CONSTRUCT statements.

Next, we turn to the use of HEX aggregate predicates in order to translate aggregate terms. Let $Q = (R, P, DS)$ and $a = agg\,(V\!:P_a)$ – here, $V \subseteq vars(P_a)$ is the tuple of variables we want to aggregate over – be an aggregate term appearing either in R or in a filter expression in P. Then, the idea is that a can be translated by an external atom $agg[aux, \overline{vars}(P_a)'[V/mask]\,](v_a)$ where

(i) $\overline{vars}(P_a)'$ is obtained from $\overline{vars}(P_a)$ by removing all variables which are not aggregated over and only appear in P_a but not elsewhere in P, i.e., from $vars(P_a) \cap vars(P) \cup V$
(ii) the variable v_a takes the place of a,
(iii) aux_a is a new predicate defined by a rule: $aux_a(\overline{vars}(P_a)') \leftarrow answer_a(\overline{vars}(P_a))$.
(iv) $answer_a$ is the predicate defining the solution set of the query $Q_a = (vars(P_a), P_a, DS)$

Example 8. The following rules mimic the CONSTRUCT query of Example 4:

```
triple(P,os:latestRelease,V_a) :- MAX[aux_a,P,R,mask](V_a),
                                  triple(P,rdf:type,doap:Project,gr).
aux_a(P,R,V) :- answer_a(P,R,V).
answer_a(P,R,V) :- triple(P,doap:release R,def), triple(R,doap:revision,V,def).
```

With the extensions the translation in [22] outlined here for extended datasets, aggregate and built-in terms we can define the solution tuples of an SPARQL++ query $Q = (R, P, DS)$ over an extended dataset now as precisely the set of tuples corresponding to the cautious extension of the predicate $\texttt{answer}_q$.

4.5 Adding Ontological Inferences by Encoding ρdf$^-$ into SPARQL

Trying the translation sketched above on the query in Example 6 we observe that we would not obtain any answers, as no triples in the dataset would match the triple pattern `?P rdf:type foaf:Agent` in the WHERE clause. This still holds if we include the vocabulary definition of FOAF at `http://xmlns.com/foaf/spec/index.rdf` to the dataset, since the machinery introduced so far could not draw any additional inferences from the triple `foaf:maker rdfs:range foaf:Agent` which would be necessary in order to figure out that Jean Deau is indeed an agent. There are several open issues on using SPARQL on higher entailment regimes than simple RDF entailment which allow such inferences. One such problem is the presences of infinite axiomatic triples in RDF semantics or several open compatibility issues with OWL semantics, see also [9]. However, we would like to at least add some of the inferences of the RDFS semantics. To this end, we will encode a small but very useful subset of RDFS, called ρdf [20] into the extended dataset. ρdf, defined by Muñoz et al., restricts the RDF vocabulary to its essentials by only focusing on the properties rdfs:subPropertyOf, rdfs:subClassOf,rdf:type, rdfs:domain, and rdfs:range, ignoring other constituents of the RDFS vocabulary. Most importantly, Muñoz et al. prove that (i) ρdf entailment corresponds to full RDF entailment on graphs not mentioning RDFS vocabulary outside ρdf, and (ii) that ρdf entailment can be reduced to five axiomatic triples (concerned with reflexivity of the subproperty relationship) and 14 entailment rules. Note that for graphs which do not mention subclass or subproperty relationships, which is usually the case for patterns in SPARQL queries or the mapping rules we encode here, even a reflexive-relaxed version of ρdf that does not contain any axiomatic triples is sufficient. We can write down all but one of the entailment rules of reflexive-relaxed ρdf as CONSTRUCT queries which we consider implicitly present in the extended dataset:

```
CONSTRUCT {?A :subPropertyOf ?C} WHERE{?A :subPropertyOf ?B ?B :subPropertyOf ?C.}
CONSTRUCT {?A :subClassOf ?C} WHERE { ?A :subClassOf ?B. ?B :subClassOf ?C. }
CONSTRUCT {?X ?B ?Y}        WHERE { ?A :subPropertyOf ?B. ?X ?A ?Y. }
CONSTRUCT {?X rdf:type ?B} WHERE { ?A :subClassOf ?B. ?X rdf:type ?A. }
CONSTRUCT {?X rdf:type ?B} WHERE { ?A :domain ?B. ?X ?A ?Y. }
CONSTRUCT {?Y rdf:type ?B} WHERE { ?A :range ?B.  ?X ?A ?Y. }
CONSTRUCT {?X rdf:type ?B} WHERE { ?A :domain ?B. ?C :subPropertyOf ?A. ?X ?C ?Y.}
CONSTRUCT {?Y rdf:type ?B} WHERE { ?A :range ?B.  ?C :subPropertyOf ?A. ?X ?C ?Y.}
```

There is one more entailment rule for reflexive-relaxed ρdf concerning that blank node renaming preserves ρdf entailment. However, it is neither straightforwardly possible, nor desirable to encode this by CONSTRUCTs like the other rules. Blank node renaming might have unintuitive effects on aggregations and in connection with OPTIONAL queries. In fact, keeping blank node identifiers in recursive CONSTRUCTs after standardizing apart is what keeps our semantics finite, so we skip this rule, and call the resulting ρdf fragment encoded by the above CONSTRUCTs ρdf^-. Some care is in order concerning strong safety of the resulting dataset when adding ρdf^-. To still ensure strong safety of the translation, we complete the predicate-class-dependency graph by additional edges between all pairs of resources connected by subclassOf or subPropertyOf, domain, or range relations and checking the same safety condition as before on the graph extended in this manner.

4.6 Implementation

We implemented a prototype of a SPARQL++ engine based on on the HEX-program solver dlvhex.[20] The prototype exploits the rewriting mechanism of the dlvhex framework, taking care of the translation of a SPARQL++ query into the appropriate HEX-program, as laid out in Section 4.4. The system implements external atoms used in the translation, namely (i) the RDF atom for data import, (ii) the aggregate atoms, and (iii) a string concatenation atom implementing both the $CONCAT$ function and the SK atom for bNode handling. The engine can directly be fed with a SPARQL++ query. The default syntax of a dlvhex results corresponds to the usual answer format of logic programming engines, i.e., sets of facts, from which we generate an XML representation, which can subsequently be transformed easily to a valid RDF syntax by an XSLT to export solution graphs.

5 Related Work

The idea of using SPARQL CONSTRUCT queries is in fact not new, even some implemented systems such as TopBraid Composer already seem to offer this feature,[21] however without a defined and layered semantics, and lacking aggregates or built-ins, thus insufficient to express mappings such as the ones studied in this article.

Our notion of extended graphs and datasets generalizes so-called networked graphs defined by Schenk and Staab [26] who also use SPARQL CONSTRUCT statements as rules with a slightly different motivation: dynamically generating views over graphs. The authors only permit bNode- and built-in free CONSTRUCTs whereas we additionally allow bNodes, built-ins and aggregates, as long as strong safety holds which only restricts recursion over value-generating triples. Another differenece is that their semantics bases on the well-founded instead of the stable model semantics.

PSPARQL [1], a recent extension of SPARQL, allows to query RDF graphs using regular path expressions over predicates. This extension is certainly useful to represent mappings and queries over graphs. We conjecture that we can partly emulate such path expressions by recursive CONSTRUCTs in extended datasets.

As an interesting orthogonal approach, we mention iSPARQL [17] which proposes an alternative way to add external function calls to SPARQL by introducing so called

[20] Available with dlvhex on `http://www.kr.tuwien.ac.at/research/dlvhex/`

[21] `http://composing-the-semantic-web.blogspot.com/2006/09/ontology-mapping-with-sparql-construct.html`

virtual triple patterns which query a "virtual" dataset that could be an arbitrary service. This approach does not need syntactic extensions of the language. However, an implementation of this extension makes it necessary to know upfront which predicates denote virtual triples. The authors use their framework to call a library of similarity measure functions but do not focus on mappings or CONSTRUCT queries.

As already mentioned in the introduction, other approaches often allow only mappings at the level of the ontology level or deploy their own rules language such as SWRL [16] or WRL [7]. A language more specific for ontology mapping is C-OWL [3], which extends OWL with bridge rules to relate ontological entities. C-OWL is a formalism close to distributed description logics [2]. These approaches partially cover aspects which we cannot handle, e.g., equating instances using owl:sameAs in SWRL or relating ontologies based on a local model semantics [14] in C-OWL. None of these approaches though offers aggregations which are often useful in practical applications of RDF data syndication, the main application we target in the present work. The Ontology Alignment Format [10] and the Ontology Mapping Language [25] are ongoing efforts to express ontology mappings. In a recent work [11], these two languages were merged and given a model-theoretic semantics which can be grounded to a particular logical formalism in order to be actually used to perform a mediation task. Our approach combines rule and mapping specification languages using a more practical approach than the above mentioned, exploiting standard languages, ρdf and SPARQL. We keep the ontology language expressivity low on purpose in order to retain decidability, thus providing an executable mapping specification format.

As a final remark, let us emphasize that our translation is based on the set-based semantics of [21,22] whereas the algebra for SPARQL defined in the latest candidate recommendation [24] defines a multiset semantics. An extension of our translation towards this multiset semantics is described in [23].

6 Conclusions and Further Work

In this paper we have demonstrated the use of SPARQL++ as a rule language for defining mappings between RDF vocabularies, allowing CONSTRUCT queries — extended with built-in and aggregate functions — as part of the dataset of SPARQL queries. We mainly aimed at setting the theoretical foundations for SPARQL++. Our next steps will involve to focus on scalability of our current prototype, by looking into how far evaluation of SPARQL++ queries can be optimized, for instance, by pushing query evaluation from our dlvhex as far as possible into more efficient SPARQL engines or possibly distributed SPARQL endpoints that cannot deal with extended datasets natively. Further, we will investigate the feasibility of supporting larger fragments of RDFS and OWL. Here, caution is in order as arbitrary combininations of OWL and SPARQL++ involve the same problems as combining rules with ontologies (see[9]) in the general case. We believe that the small fragment we started with is the right strategy in order to allow queries over networks of lightweight RDFS ontologies, connectable via expressive mappings, which we will gradually extend.

References

1. Alkhateeb, F., Baget, J.-F., Euzenat, J.: Extending SPARQL with Regular Expression Patterns. Tech. Report 6191, Inst. National de Recherche en Informatique et Automatique (May 2007)
2. Borgida, A., Serafini, L.: Distributed Description Logics: Assimilating Information from Peer Sources. Journal of Data Semantics 1, 153–184 (2003)
3. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: Contextualizing Ontologies. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, Springer, Heidelberg (2003)
4. Chen, W., Kifer, M., Warren, D.: HiLog: A Foundation for Higher-Order Logic Programming. Journal of Logic Programming 15(3), 187–230 (1993)
5. de Bruijn, J., Franconi, E., Tessaris, S.: Logical Reconstruction of Normative RDF. In: OWLED 2005. OWL: Experiences and Directions Workshop, Galway, Ireland (2005)
6. de Bruijn, J., Heymans, S.: A Semantic Framework for Language Layering in WSML. In: RR2007. First IntĺConf. on Web Reasoning and Rule Systems, Innsbruck, Austria (2007)
7. de Bruijn, J(eds.): Web Rule Language (WRL), W3C Member Submission (2005)
8. Decker, S., et al.: TRIPLE - an RDF Rule Language with Context and Use Cases. In: W3C Workshop on Rule Languages for Interoperability, Washington D.C., USA (April 2005)
9. Eiter, T., Ianni, G., Polleres, A., Schindlauer, R., Tompits, H.: Reasoning with Rules and Ontologies. In: Reasoning Web 2006, pp. 93–127. Springer, Heidelberg (2006)
10. Euzenat, J.: An API for Ontology Alignment. In: Proc. 3rd International Semantic Web Conference, Hiroshima, Japan, pp. 698–712 (2004)
11. Euzenat, J., Scharffe, F., Zimmerman, A.: Expressive Alignment Language and Implementation. Project Deliverable D2.2.10, Knowledge Web NoE (EU-IST-2004-507482) (2007)
12. Faber, W., Leone, N., Pfeifer, G.: Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. In: Alferes, J.J., Leite, J.A. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, Springer, Heidelberg (2004)
13. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9, 365–385 (1991)
14. Ghidini, C., Giunchiglia, F.: Local model semantics, or contextual reasoning = locality + compatibility. Artificial Intelligence 127(2), 221–259 (2001)
15. Hayes, P.: RDF Semantics. Technical Report, W3C, W3C Recommendation (February 2004)
16. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission (2004)
17. Kiefer, C., Bernstein, A., Lee, H.J., Klein, M., Stocker, M.: Semantic Process Retrieval with iSPARQL. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, Springer, Heidelberg (2007)
18. Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-oriented and Frame-based Languages. Journal of the ACM 42(4), 741–843 (1995)
19. Malhotra, A., Melton, N. W.J.: (eds.) XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Recommendation (January 2007)
20. Muñoz, S., Pérez, J., Gutierrez, C.: Minimal Deductive Systems for RDF. In: ESWC 2007. 4th European Semantic Web Conference, Innsbruck, Austria (2007)
21. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 30–43. Springer, Heidelberg (2006)
22. Polleres, A.: From SPARQL to Rules (and back). In: WWW 2007. 16th World Wide Web Conference, Banff, Canada (May 2007)

23. Polleres, A., Schindlauer, R.: dlvhex-sparql: A SPARQL-compliant Query Engine based on dlvhex. In: ALPSWS 2007. 2nd Int. Workshop on Applications of Logic Programming to the Web, Semantic Web and Web Services, Porto, Portugal (2007)
24. Prud'hommeaux, E., Seaborne, A.:(eds.) SPARQL Query Language for RDF. W3C Candidate Recommendation (June 2007)
25. Scharffe, F., de Bruijn, J.: A Language to specify Mappings between Ontologies. In: IEEE SITIS2005. First Int. Conf. on Signal-Image Technology and Internet-Based Systems
26. Schenk, S., Staab, S.: Networked rdf graphs. Tech. Report, Univ. Koblenz (2007), `http://www.uni-koblenz.de/~sschenk/publications/2006/ngtr.pdf`
27. Schindlauer, R.: Answer-Set Programming for the Semantic Web. PhD thesis, Vienna University of Technology (December 2006)
28. Ullman, J.: Principles of Database & Knowledge Base Systems. Comp.Science Press (1989)

OntoPath: A Language for Retrieving Ontology Fragments

E. Jiménez-Ruiz[1], R. Berlanga[1], V. Nebot[1], and I. Sanz[2]

[1] Departamento de Lenguajes y Sistemas Informáticos
[2] Departamento de Ingeniería y Ciencia de los Computadores
Universitat Jaume I (Castellón)
{ejimenez, victoria.nebot, berlanga, isanz}@uji.es

Abstract. In this work we introduce a novel retrieval language, named *OntoPath*, for specifying and retrieving relevant ontology fragments. This language is intended to extract customized self-standing ontologies from very large, general-purpose ones. Through *OntoPath,* users can specify the desired detail level in the concept taxonomies as well as the properties between concepts that are required by the target applications. The syntax and aims of OntoPath resemble XPath's in that they are simple enough to be handled by non-expert users and they are designed to be included in other XML-based applications (e.g. transformations sheets, semantic annotation of web services, etc.). *OntoPath* has been implemented on the top of the graph-based database *G*, for which a Protégé OWL plug-in has been designed to access and retrieve ontology fragments.

Keywords: Query Languages, Ontologies, Semantic Web, Ontology Fragments.

1 Introduction

The Semantic Web is intended to extend the current web by creating knowledge objects that allow both users and programs to better exploit the huge amount of resources. The cornerstone of the Semantic Web is the definition of widely agreed ontologies conceptualizing the knowledge behind web resources. Nowadays, several efforts are focused on building very large ontologies that are continuously growing as new knowledge is added to them by the respective communities. This happens mainly in the biomedical domain (e.g. GO[1], GALEN[2], FMA[3], NCI-Thesurus[4], Tambis[5], BioPax[6], etc).

[1] Gene Ontology: http://www.geneontology.org/
[2] Galen Ontology: http://www.opengalen.org/
[3] Foundational Model of Anatomy: http:// fma.biostr.washington.edu/
[4] NCI taxonomy/ontology: http://nciterms.nci.nih.gov/NCIBrowser
[5] Tambis Ontology: http://www.cs.man.ac.uk/~stevensr/tambis-oil.html
[6] BioPax Ontology: http://www.biopax.org/

However, the very large size of these ontologies as well their variety makes it difficult to deploy them in particular applications. The first problem is scalability. Most of these ontologies are expressed in OWL-DL, with different degrees of expressivity. The classification of new concepts, queries and assertions require the use of reasoners (e.g. Pellet[7], Racer[8], etc.), but they are not able to handle even medium-size ontologies [1, 2]. A second problem is the application of these ontologies to concrete applications. Such an application usually does not require comprehensive descriptions of the domains but rather a handful subset of concepts and properties from them (i.e. a local view of the domain [3]). Another important issue is their visualization in ontology editors, in which large ontologies have difficulties to be loaded and properly displayed.

This paper presents a new query language, *OntoPath*, for retrieving consistent fragments from domain ontologies. It is based on the XPath [24] syntax but it is applied over ontologies described in OWL. In contrast to other Semantic Web query languages such as SparQL [4] and RQL [5], this language is intended to extract the necessary knowledge to build new ontologies according to user and application requests. As a consequence, query results are also OWL files.

OntoPath is being applied within the Health-e-Child project [25] as a mechanism to explore large biomedical ontologies and also to perform the vertical integration of the biomedical knowledge via the definition of ontology fragments and connections between them (i.e. semantic bridges) [6]. Another direct application of *OntoPath* is the extraction of analysis dimensions for building OLAP-based multidimensional models for biomedical research [7]. Finally, it is also being applied to the compositions of Semantic Web Services. In this case, semantic services are represented as concepts and their input/output parameters as their properties. Service chains can be then obtained through Ontopath queries [8].

The rest of the paper is organized as follows. Section 2 gives a brief review of related works. An overview of the proposal is presented in Section 3. Section 4 describes how ontologies are parsed and stored into the semi-structured database G. OntoPath syntax and semantics are presented in Section 5. Section 6 is dedicated to OntoPath query processing. Experiments and examples are presented in Section 7. Finally, Section 8 gives some conclusions and future work.

2 Related Work: Ontology Modularization and Querying

The necessity of working with ontology modules is well known in the literature and several approaches can be found to modularize and query ontologies. In [1] a good review of ontology modularization formalisms is presented, among them: Distributed Description Logic [9], Modular Reuse of Ontologies [10], Package Based DL [11] and Network Partitioning [12]. Although these works propose good formalisms to automatically define ontology modules and their connections, they do not provide mechanisms to guide the modularization, that is, users do not participate in the partitioning of the ontology.

[7] Pellet reasoner: http://pellet.owldl.com/

[8] RacerPro reasoner: http://www.racer-systems.com/

Alternatively to these approaches, traversal-based ones are aimed at extracting ontology fragments according to user preferences, usually specified as a set of concepts and properties of interest. PROMPT [13], MOVE [14] and OntoPathView [18] are examples of traversal-based methods. All of them deal with pure frame-based ontologies. This is a serious limitation since most ontologies are being expressed in the standard language OWL, which assumes as underlying model the Description Logic (DL).

Recently, Seidenberg and Rector [2] have proposed a segmentation algorithm to extract fragments (i.e. segments) from large ontologies in OWL, more specifically the GALEN ontology. This algorithm resembles traversal-based approaches in that it starts from a concept of interest and then navigates through the concept hierarchy to identify not only related properties and concepts but also any element necessary to describe and classify them. Thus, their main aim is to keep as much as possible of the inference capabilities of the original ontology in the resulting segments. However, resulting segments use to be very large, some of them still intractable by current editors and reasoners. This is because affected axioms can be scattered across the whole concept hierarchy.

In this paper, we present a different approach for extracting fragments from OWL ontologies. Our method differs from the previous one in that we do not preserve all the axioms that are necessary to classify the extracted concepts. Instead, our approach makes explicit much of this knowledge in the resulting fragments. In other words, instead of generating new OWL with implicit knowledge to be inferred, we generate specific and explicit fragments that will not require reasoning capabilities. As a consequence, much smaller and specific fragments can be obtained. This is especially useful when generating OWL fragments that will be used in specific final applications (e.g. a data warehouses).

It is worth mentioning that there exist other query languages for the Semantic Web, such as RQL [5], SparQL [4], KAON views [15] and RVL [23]. However, these languages deal with RDF/S ontologies, which are less expressive than OWL ones. Moreover, their aim is not to build new closed and consistent ontologies as ours, but providing relational-like data satisfying user requests (e.g. all elements with a specific date and author).

3 Ontology Management System Architecture

The development of OntoPath has been carried out in the context of a new database management system aimed at storing, organising and retrieving customized pieces of knowledge to be used in specific applications [16]. This system relies on the semi-structured database G[9] and the language OWL. The former was chosen because of its great capability for storing and retrieving large graph-like data. Figure 1 summarizes the system architecture, whose main modules are described in turn.

[9] A. Rios ”G Platform (Helide S.A)”: http://www.helide.com, http://www.maat-g.com

- *OWL parser*[10] *and builder*. The former consists of an ad-hoc SAX-based parser, which builds the necessary objects from an input ontology to make it persistent in the database. The latter allows OWL files to be generated from database objects.

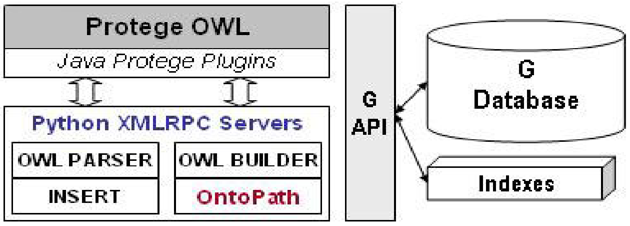


Fig. 1. Architecture of the Ontology Management System

- *G database*. It has been used as a backend to store, index and retrieve the OWL ontologies as graphs. To store ontologies four object types are defined, namely: *ontology*, *property*, *concept*, and *enumeration*; the latter for nominal lists. Figure 2 summarizes the four object types and the existing references between them. All OWL constructors have been regarded so that ontologies can be loaded and retrieved without semantic lost.

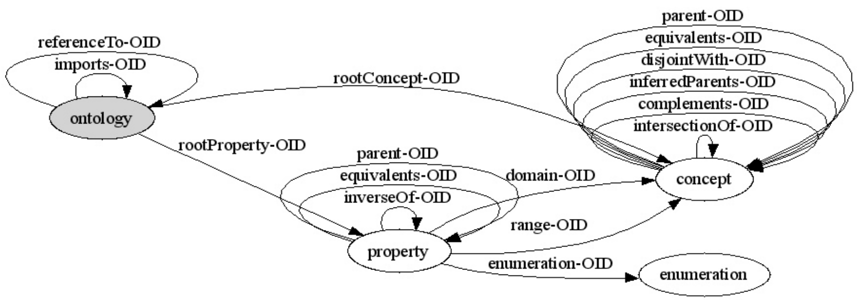


Fig. 2. Meta-schema for storing OWL Ontologies into G

- *OntoPath*. It is the proposed query language for retrieving ontology fragments (personalized modules or views) from the domain ontologies. By using OntoPath, users can specify the desired detail level in the concept taxonomies as well as the properties that are required by the target applications. OntoPath is the main focus of this work.
- *Protégé-OWL*. It is the front-end of the system, which allow users to visualize and edit the stored ontologies (fragment or whole ones). A specific plug-in has been recently designed to create and browse ontology modules and fragments [19].

[10] Python OWL Parser: http://krono.act.uji.es/people/Ernesto/Owl_Parser/

4 Ontology Parsing and Storage

From now on, we assume that ontologies are expressed in OWL-DL 1.0 (which uses a subset of the DL $\mathcal{SHOIN}$(D) [20]), and therefore we use standard Description Logic (DL) syntax when discussing semantic issues in the paper.

As previously mentioned, ontologies are parsed and stored in the G database as graphs (see Figure 2), where nodes represent ontology entities (e.g. concepts, properties and nominals) and edges between nodes represent their different relationships (e.g. *subClassOf*, *domain*, *range*, *intersectionOf*, etc.) From now on, we represent G database records as follows:

$$R= Type(att_1=v_1,\ldots,att_n=v_n)$$

Where *Type* is the type of the record (e.g. *ontology*, *concept*, etc.) , and $att_i=v_i$ states that the record attribute att_i takes the value v_i. Multi-valued attributes are also allowed, taking the form $att_i=v_1,..,v_k$. In order to support the heterogeneity of the database records, attributes lists associated to a record type can be of arbitrary length (i.e. optional attributes are allowed) and formats (i.e. an attribute can take values from different data types at each record). Additionally, values can be references to other database records. For this purpose, record unique identifiers are used. As an example, the following records represent a simple ontology with two concepts and one property:

O=*ontology*(*name*='Simple.owl', rootConcept=C1, rootProperty=P1)
*C*1=*concept*(*name*='Thing')
*P*1= *property* (*name*='PropertyThing')
*C*2=*concept*(*name*='Person', *subClassOf*=*C*1)
P2= *property*(*name*='hasFriend', *range*=*C*2, *domain*=*C*2, *subPropertyOf*=*P*1)

P1, P2, *C*1, *C*2 and *O* are the unique identifiers of the corresponding data records. To denote an attribute *att* of a record *R*, we use *R.att*, for example C2.*subClassOf*.

As explained in next sections, *OntoPath* query processing relies on traversing both the concept hierarchy and the association graph between concepts. The former consists of the links present at *subClassOf* record attributes, whereas the latter consists of the property records linking concepts through its domains and ranges.

As OWL properties can appear in many different contexts (i.e. class restrictions), we introduce another record type named inferred property (*i-property*). An *i-property* record has a similar structure to original *property* records, but its interpretation must be restricted to the context it happens. For example, considering the class definition $C \sqsubseteq A \sqcap \exists R.B$, the record *i-property*(*name=R*, *domain=C*, *range=B*) means that in the context of *C*, some instances of *A* are related through *R* to some instances of *B*. For this reason, *i-property* records will have attached the DL expression from which they where extracted.

Ontology instances (i.e. individuals in DL jargon) are also stored as database records, preserving their references through database object references. For example, the following instances belong to the previous ontology:

I1=instance(name='John', hasFriend=I2, type=C2)
I2= instance(name='Peter', hasFriend=I1, type=C2)

Following subsections describe how complex DL expressions resulted from OWL constructors can be approximated to enrich as much as possible database records, and therefore to make explicit the necessary knowledge for solving *OntoPath* queries.

4.1 Approximating DL Expressions

As mentioned in the introduction, large ontologies cannot be classified by current DL reasoners due to their inherent exponential complexity. This means that concepts that are not explicitly related to others cannot be properly classified, and will be unconnected to the rest of the ontology. However, there are some simple and frequent patterns appearing in OWL definitions that can be treated without requiring tableau-like reasoning. The application of these patterns establishes further relations between concepts and properties in the database, and they will provide a more complete vision of the ontology as a graph.

The current list of patterns is as follows:

1. **Inferred parents:** if a concept is defined as an intersection (both equivalent and a subset) of a set of concepts, we can infer that it is a subclass of them.

   $C \equiv C_1 \sqcap \ldots \sqcap C_n \Rightarrow C.\mathit{subClassOf}{=}C_1,\ldots,C_n$
2. **Inferred children:** if a class is defined as the *equivalent* of the union of a set of classes, these classes are its subclasses.

   $C \equiv C_1 \sqcup \ldots \sqcup C_n \Rightarrow C_1.\mathit{subClassOf.append}(C)$, …, $C_n.\mathit{subClassOf.append}(C)$
3. **Inferred domains:** If a restriction of the form $\exists R.D$ or $\forall R.D$ is found as a class definition, the corresponding property record is build:

   $C \sqsubseteq \exists R.D \Rightarrow \mathit{i\text{-}property}(\mathit{name}{=}R, \mathit{domain}{=}C, \mathit{range}{=}D)$.

   Both *C* and *D* must be named classes. Notice that this pattern is also applied to cardinalities, but the i-property is created without a defined range (qualified cardinalities are not considered).
4. **Creation of new classes:** If an axiom of the form $C \sqsubseteq \exists R.(D \sqcap \exists S.E)$ is found, being *C* and *D* named classes, then a new named class *D'* ($D' \equiv D \sqcap \exists S.E$) is created, with *D'.subClassOf=D*. Notice that by applying pattern 3, the following property record is also created: *i-property*(*name=R*, *domain=C*, *range=D'*). Finally, the pattern 3 is recursively applied to the sub-expression ($\exists S.E$). New class names are generated with the elements of the expression: *D'.name=D*_with_*S*_*E*.
5. **Nominals:** the occurrence of nominals is treated as a special case of pattern 3 if they appear inside a restriction: *allValuesFrom*, *someValuesFrom* and *hasValue* cases. The correspondent *i-property* record is created (inferred domain), an also an *enumeration* record for the set of nominals. Notice that for *hasValue* cases ($\ni$) the enumeration record will have only one instance value associated.

   $C \sqsubseteq \exists R.\{i_1, i_2 \ldots, i_n\} \Rightarrow \mathit{i\text{-}property}(\mathit{name}{=}R, \mathit{domain}{=}C, \mathit{enumeration}{=}E)$.

   $C \sqsubseteq \ni R.\{ i_1\} \Rightarrow \mathit{i\text{-}property}(\mathit{name}{=}R, \mathit{domain}{=}C, \mathit{enumeration}{=}E)$.

   $E{=}\mathit{enumeration}(\mathit{name}{=}R\text{-}\mathit{nominals}, \mathit{list\text{-}of\text{-}values}{=}i_{1,\ldots}, i_n)$

These patterns are applied when parsing the OWL file. For each class definition, a DL expression is built from the corresponding OWL constructors, and it is normalized (conjunctive form) before applying the patterns. These DL expressions are also stored in the database records of their classes.

In order to better understand the proposed patterns, let's consider the following class definitions examples in DL. The first one has been extracted from the Pizza[11] ontology and the second one has been taken from a portion of the GALEN ontology expressed in OWL[12]:

Pizza Example

$SpicyPizza \equiv Pizza \sqcap \exists hasTopping.(PizzaTopping \sqcap \exists hasSpiciness.Hot)$

By applying the previous patterns, the following records are generated; notice that a new class has been created (*PizzaTopping_with_ hasSpiciness_Hot*) and new explicit relationships are stored (i-properties *i-hS* and *i-hT*):

```
P = concept(name=Pizza, ....)
SP = concept(name=SpicyPizza, subClassOf=P)
PT = concept(name=PizzaTopping, ...)
H = concept(name=Hot,...)
PTH = concept(name= PizzaTopping_with_ hasSpiciness_Hot, subClassOf=PT)
hT = property(name=hasTopping, ...)
hS = property(name=hasSpiciness, ...)
i-hT = i-property(name=hasTopping, subPropertyOf=hT, domain=SP, range=PTH)
i-hS = i-property(name=hasSpiciness, subPropertyOf=hS, domain=PTH, range=H)
```

Figure 3 shows the representation of the previous registers with all the DL constructors represented explicitly in the graph by means of relationships between concepts

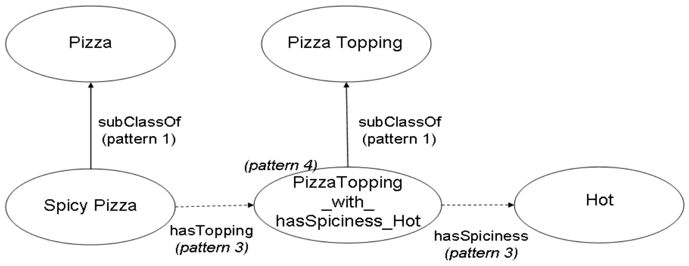


Fig. 3. Graph representation for Pizza example

Let us emphasize that the creation of class *PizzaTopping_with_ hasSpiciness_Hot* (by means of pattern 4) is also proposed in the Pizza-OWL Tutorial[13] in which they create manually the class *HotPizzaTopping*.

[11] Pizza OWL: http://www.co-ode.org/ontologies/pizza/
[12] Galen ontology in OWL: http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/galen.owl
[13] Pizza-OWL Tutorial: http://www.co-ode.org/resources/tutorials/protege-owl-tutorial.php

Galen Example

$$AcuteIschaemicCardiacPathology \equiv CardiacPathology \sqcap \exists isConsequenceOf.(Ischaemia \sqcap \exists hasChronicity.(Chronicity \sqcap \exists hasState.Acute))$$

For the previous DL expression two classes are created by applying pattern 4 recursively (*Ischaemia_with_hasChronicity_Chronicity_with_hasState_Acute* and *Chronicity_with_hasState_Aute*). Next we present the created records for the DL expression:

```
CP = concept(name= CardiacPathology, ....)
AICP = concept(name= AcuteIschaemicCardiacPathology, subClassOf=CP...)
I = concept(name= Ischaemia, ...)
C = concept(name= Chronicity,...)
A = concept(name= Acute,...)
ICA=.concept(subClassOf=I, name= Ischaemia_with_hasChronicity_Chronicity_with_hasState_Acute, ... )
CA = concept(name=Chronicity_with_hasState_Aute, subClassOf=C)
iCO = property(name=isConsequenceOf, ...)
hC = property(name=hasChronicity, ...)
hS = property(name=hasState, ...)
i-iCO = i-property(name=isConsequenceOf, subPropertyOf=iCO, domain=AICP, range=ICA, ...)
i-hC = i-property(name=hasChronicity, subPropertyOf=hC, domain= ICA, range=CA ...)
i-hS = i-property(name=hasState, subPropertyOf=hS, domain=CA, range=A,...)
```

Finally, it is worth mentioning that for the Galen portion (3002 classes and 413 properties) are generated more than two hundred classes following pattern 4 and near two thousand inferred properties following patterns 3 and 5.

4.2 Expressivity of Stored and Retrieved Ontologies

During the storage of an ontology, a DL reasoner can be applied to classify all its concepts and instances according to the logic subsumption relationship. However, if such a reasoner is not available or it cannot be applied to the ontology due to its size, relations between concepts will be only approximate as some subsumption relationships cannot be detected. This is especially critical when union, negations or complex axioms (i.e. $C \sqsubseteq \exists R.D \sqcup \exists S.E \sqcup (F \sqcap \neg G)$) are included in the concept definitions, as these constructors can only be treated under very strict conditions. In these cases, a set of anonymous classes are created to maintain the graph structure, and the original DL expressions are kept in the concept records in order to be reconstructed when required.

Next table summarizes the treatment given in the storage for each class or role constructor. Notice that, OWL DL ontologies using constructors that are not treated will produce approximate answers (i.e. possibly incomplete) to OntoPath queries. Fortunately, many domain ontologies seldom use negation and union in the concept definitions.

Table 1. Treated DL constructors

Constructors	Treatment
$\mathcal{AL}$ - ∀R.C - Universal Restrictions	Pattern 3
$\mathcal{AL}$ - Concept Intersection	Pattern 1
$\mathcal{E}$ - ∃R.C - Existential Qualifications	Pattern 3
$\mathcal{C}$ - Complex Concept Negation	Not treated.
$\mathcal{U}$ - Concept union	Pattern 2, only the case of concept equivalence.
$\mathcal{H}$ - Role Hierarchy	Stored role attribute *subPropertyOf*
$\mathcal{R}$ - Complex Role Inclusion Axioms	Not considered for OWL 1.0
$\mathcal{I}$ - Inverse Properties	Stored role attribute *inverseOf*
$\mathcal{O}$ - Nominals	Pattern 5
$\mathcal{N}$ - Cardinality Restrictions	Pattern 4
$\mathcal{Q}$ - Qualified Cardinality Restrictions	Not considered for OWL 1.0
(D) - Data Type Properties	Stored role attribute *PropertyType*

5 OntoPath Queries

An OntoPath query is a tree relating concepts and properties of the ontology. As in XPath, trees are expressed through paths with predicates that impose conditions through the traversed nodes. However, unlike XPath (and SparQL), the answer to an OntoPath query is not a set of pointers to the nodes in the graph satisfying the specified conditions, but a consistent and closed fragment of the ontology (i.e. a sub-graph) containing the required concepts and properties. Consistent means that all concepts in the fragment are satisfiable, whereas closed means that the fragment includes all the necessary concepts and properties to make it consistent. Notice that closeness and consistency may involve changes over the original knowledge, mainly over frontier concepts of the ontology fragment, in which a partial definition of them has been extracted. Nevertheless, the main objective is to define a party's view of a domain [3].

5.1 Query Expressions

An OntoPath query contains two types of tree nodes: concept nodes and property nodes. A property node have only one child concept node, whereas a concept node can have more than one child property node. Tree nodes also contain predicates that must be satisfied in the retrieved fragment. Figure 4 shows the interpretation of a simple OntoPath query like Disease/related-to/Gene, where an ontology fragment is extracted with diseases (if exist) directly related by means of the related-to property to the Gene concept.

The simplest query in OntoPath consists of specifying a concept *C*. The result consists of an ontology fragment containing all their sub-concepts (i.e. the concept taxonomy under *C*) an all its instances. No object properties are retrieved nor relations between instances involved in the fragment. Data type properties associated to these concepts (direct and inferred ones) are always included in the fragment.

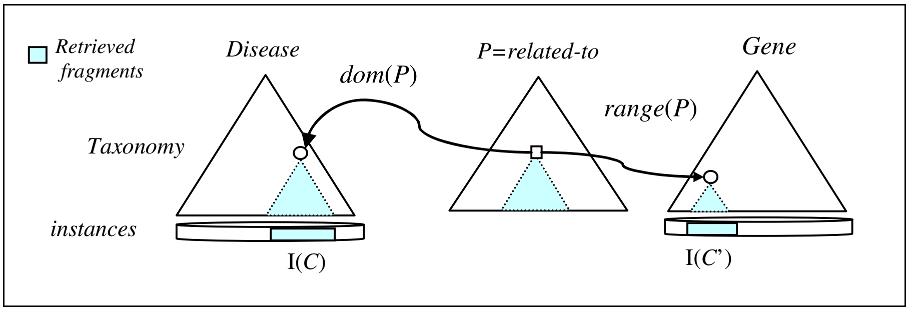


Fig. 4. Graphical interpretation of the *OntoPath* query Disease/related-to/Gene

Let *sub*(C) denote the set of all the sub-concepts of C, and *sub*(*P*) be the set of all the sub-properties of *P* and their respective *i-property* records. To state aggregation conditions over concepts, we use the syntax: *C*/*P*/*C*', where *C* and *C*' represent concepts, and *P* denotes a property. A fragment satisfying this query contains those sub-concepts of *C* and *C*', denoted R(*C*) and R(*C*') respectively, as well as those sub-properties of *P*, denoted R(*P*), such that:

- $R(C) \subseteq sub(C)$, $R(C') \subseteq sub(C')$ and $R(P) \subseteq sub(P)$
- $\forall p \in R(P)$, $dom(P) \in R(C)$ and $range(P) \in R(C')$
- $\forall c \in R(C)$, $\exists p \in R(P)$ such that $c = dom(p)$
- $\forall c \in R(C')$, $\exists p \in R(P)$ such that $c = range(p)$

We denote with I(*C*) to the direct instances (individuals) of *C*. Thus, the query *C*/*P*/*C*' also retrieves all the direct instances of all concepts in *sub*(*C*) and *sub*(*C*'). Notice that we can extend the previous query definition to paths of any length: $C_1/P_1/C_2/P_2/C_3/P_3/\ldots$

Query expressions can contain the symbol "*" to denote the concept ⊤(*Thing*), and the symbol "?" to denote any property (i.e. *sub*(?) denotes all the properties in the ontology).

Some examples of OntoPath queries are the following ones:

- RheumatoidArthritis: The taxonomy and instances below RA diseases are retrieved.
- Disease/?/RheumatoidFactor: Returns a fragment with diseases (if exist) directly related by means of any property to Rheumatoid Factor.
- AutoinmuneDisease/?/*/?/GenePTPN22: Returns a fragment with autoimmune diseases related to gene PTPN22 along with all the concepts and properties participating in their relationships.
- RheumatoidArthritis/hasTreatment/ *: Returns a fragment with RA diseases and their treatments.

Notice that the query *C*/*P*/*C*' is equivalent to the concept $C \sqcap \exists P.C'$. Similarly, the query *C*/*P*/* is equivalent to $C \sqcap \exists P.\top$ and */*P*/*C* to $\exists P.C$. One can think that fragments for them could be obtained by including their equivalent concepts in the ontology and applying some reasoner to them. This has several limitations. Firstly, for each query we need to modify the original ontology and consequently to check the new ontology from scratch. Second, as previously mentioned reasoning is not possible for large ontologies. Finally, extracting the ontology fragment involved by a query is a difficult task from the completion graphs generated by tableau algorithms. For these reasons, we propose an ad-hoc and fast algorithm for processing OntoPath queries (see Section 6).

5.2 Predicates

As in XPath, we use square brackets [ ] to specify a predicate that must be satisfied by the involved concept or property of the query expression. If multiple conditions are required we use one []-expression for each one. They are always interpreted as a conjuction of predicates. On the contrary to XPath, we will use only []-expressions instead of the logic equivalent ones.

Twig Queries: Predicates allow us to express twig queries, that is, tree-like aggregate conditions over the concepts and properties of the query. These have the form:

$$C[P_1/C_1][P_2/C_2] \ldots$$

This is equivalent to the DL query $C \sqcap \exists P1.C1 \sqcap \exists P2.C2$.

When a twig query is required, the condition for the ontology fragment associated to each branching node (e.g. *C*) is as follows:

$$\forall c \in R(C),\ \forall P_i \in children(C),\ \exists p \in R(P_i),\ \text{ such that } c = dom(p)$$

Future extensions of *OntoPath* will include negation and disjunction in the predicates. However, such an extension requires the inclusion of reasoning mechanisms for classifying the appropriate property domains and ranges participating in the results.

Querying Metadata: These predicates are useful for selecting parts of the ontology that were created by different authors or that contain useful annotations for retrieving relevant concepts to certain applications (e.g. lexicon, keywords, etc.) They have the following syntax:

Node[@*annotation*]
Node[@*annotation* <op> *value*]

Here *Node* is either a concept or a property of the ontology. The first expression means that the specified annotation for the OWL element exists, whereas the second one indicates that the assigned value for the specified annotation must satisfy the expressed condition. For this purpose, <op> represents any binary operator over atomic data types (e.g. <, >, ==, !=, *like*, etc.) Instances including annotated elements will be included in the resulting fragment. For example, the query Disease[@source=NCI] retrieves all the disease concepts stemming from NCI thesaurus, along with the instances created for them.

Notice that reasoning over annotations is undecidable (belongs to OWL-Full category), and therefore we cannot use a reasoner to build fragments under metadata conditions.

Data filters: With this predicates, only concepts having the specified features are selected. Moreover, only instances satisfying the data filter will be included in the result. The syntax is as follows:

Concept[*Datatype_Property* <op> *value*]

Here <*op*> also represents any binary operator over atomic data types. In DL these conditions are expressed via concrete domains. However, these are not fully supported by current versions of OWL. In the future, it would be interesting to have a classification of concepts involving data type properties (for example concepts involving age intervals for disease onsets) and then filter the ontology concepts according to them (for example, selecting only concepts that involve disease onsets for infants). However, now this kind of filters can be only applied to instances that explicitly set the specified data type property.

6 Query Processing

The query processor of OntoPath has been implemented as an *interpreted grammar*. The grammar output consists of a tree where each node contains the necessary actions to retrieve the required objects from the database and to build the result set with them. All these actions require some basic query over the G database (API), see Figure 5. Due to size limitations, a full description of the underlying algebra cannot be included in the paper.

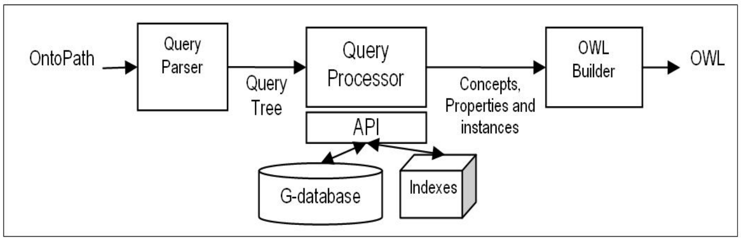


Fig. 5. Query Processing for OntoPath

Query Trees

In the rest of the section, concept nodes are denoted with *C*, *C*', etc., property nodes with *P*, *P*', etc. *parent*(X) denotes the parent node of *X* in the query tree, whereas *children*(*C*) and *child*(*P*) denote the children nodes of *C* and the unique child node of *P* respectively. The concepts or properties denoted by the query node *X* are accessed with *name*(*X*).

Our approach for query processing relies on the following principles:

- For each query concept node *C* we must keep updated the concepts in *sub*(*name*(*C*)) that covers all the sub-concepts required by the parent and children query nodes. We call it R(*C*).

- For each query property node *P* we must keep updated those properties in *sub*(*name*(*P*)) that satisfies the conditions imposed by the parent and child query nodes. We call it R(*P*).
- We call empty concept, denoted as $\perp$, to any unsatisfiable concept. Thus, if R(*C*)= $\perp$means that the query has no solution. Similarly, if some property node has R(*P*)=$\varnothing$ then the query has no solution too.
- Two basic functions are necessary to calculate R(*C*) and R(*P*), namely:
  - o *Nearest Common Descendant*: NCD(*u*, *v*) returns the nearest concept that is descendant of both *u* and *v*. It follows that NCD(*u*,*) = *u*, NCD(*, *v*)=*v*, NCD($\perp$, *v*)= $\perp$ and NCD(*u*, $\perp$)=$\perp$. If nodes *u* and *v* have not a common descendant (i.e. *sub*(*u*) and *sub*(*v*) are disjoint) then NCD(*u*, *v*)= $\perp$. This function can be defined over a set of concepts, denoted NCD(*S*), so that it returns the nearest common descendant of all the nodes in *S*.
  - o *Nearest Common Ancestor*: NCA(*u*, *v*) returns the nearest common ancestor of both *u* and *v*. It follows that NCA(*u*,*)=*, NCA(*,*v*)=*, NCA($\perp$, *v*)= $\perp$ and NCA (*u*, $\perp$)=$\perp$. This function can be defined over a set of concepts, NCA(*S*), so that it returns the nearest common ancestor of all the nodes in *S*.

The initial values of R sets are calculated as follows:

- R(*P*)=*sub*(*name*(*P*)) if P≠?. Otherwise, it is initialized taking into account the values of its parent and child nodes as follows:
  $R(P)=\{p \mid p\in sub(?) \wedge NCD(dom(p), R(parent(P)) \neq \perp \wedge$
  $NCD(range(p), R(child(P)) \neq \perp\}$
- For concept nodes C, R(*C*)={name(*C*)}, that is, they just contain the name associated to the concept specified in the query node.

R sets are updated as follows:

$$R(C)^{new} = NCD(\{NCD(dom(p),R(C)^{old})\}_{p\in \mathrm{name(P)}})\}_{P\in \mathrm{children(C)}}) \cup NCD(range(parent(C)), R(C)^{old}) \quad (1)$$

$$R(P)^{new} = \{\, p \mid p\in R(P)^{old}, NCD(dom(p),R(parent(P)) \neq \perp \wedge NCD\,(range(p),R(child(P)) \neq \perp\} \quad (2)$$

Notice that whenever a node *C* changes its R(*C*), its parent and children must be revised. Similarly, whenever a node *P* changes its R(*P*), its parent and child must be revised.

Graph Indexes

The efficient evaluation of queries requires the use of index structures. The navigation of trees and DAGs (finding ancestors, descendants and siblings) has been extensively studied in the literature; following Christophides et al.'s analysis in [21], in our implementation we have adopted a variation of Agrawal, Borgida and Jagadish's interval-based encoding technique [22].

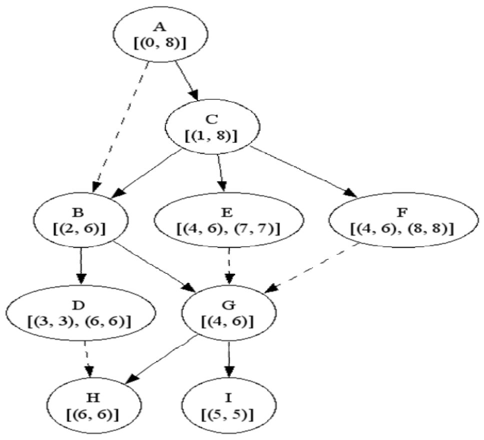


Fig. 6. Example of interval of type postorder-index assigned to a concept hierarchy

Agrawal et al's technique consists on numbering each node in the graph with integers based on a preorder and postorder traversal of the spanning tree, and then computing a set of intervals which compactly represent all the descendants of each node. This encoding is especially suited for the computation of subsumption and NCD, which are reduced to simple interval arithmetic operations. In contrast, the computation of NCA is linear in the worst case. In our tests this has not been an issue, since NCA is the least frequently used primitive; in any case, there are well-known approaches to compute NCA in constant time at little extra spatial cost, should it be deemed necessary.

6.1 Query Processing Algorithm

The query processing algorithm is as follows:

1. Initializing R sets for all query tree nodes.
2. First calculation of R sets for all query nodes. Put in a queue Q all the children and parents of the updated nodes. Finish the query processing if some concept node contains $\perp$or some property node is empty.
3. Until Q is not empty

   Pop a node from Q and revise it. That is, apply expressions (1) and (2) to update its R sets. Stop query processing if it contains $\perp$or is empty. If successfully updated then put in Q its parent and children nodes.
4. For each concept node *C*, for each concept $c \in R(C)$, construct an OWL fragment taking into account the selected properties and groups generated in the previous steps. DL expressions will be expanded only if all its contained concepts and properties are included in the fragment.
5. For each property node *P*, construct an OWL fragment taking into account the inferred domain and ranges as well as the selected property taxonomy.
6. Retrieve all the instances of all the selected concepts such that they satisfy the filter conditions stated in the corresponding concept node.

If predicates over metadata are included in the query tree, we must first proceed as follows. For each concept node *C* we must select all concept in *sub*(*C*) satisfying the predicate. We must update *sub*(*C*) and re-construct the concept taxonomy under *C* for the selected concepts. Notice that NCA and NCD functions must be now applied to the re-constructed hierarchy.

The complexity of the query processing algorithm is linear with respect to the number of property records included in the query tree. This is because each iteration in step 3 can drop at least one property record from R(*P*), and the algorithm stops when either no changes are produced or some set R(*P*) becomes empty.

7 Experiments and Results

In order to check the results obtained with *OntoPath* we have designed a set of queries over well-known ontologies in the biomedical domain, namely: GALEN (a fragment in OWL format) and NCI (version 03.09d[14]). The former is more expressive and smaller than the latter.

Tables 2 and 3 summarize the results of the retrieved fragments for several queries. These queries go from very simple taxonomy retrieval to complex twig queries with different detail levels. As it can be seen, the size of the generated fragments is relatively small (see last column), although it clearly depends on the generality of the fetched query. In the NCI ontology-thesaurus, class definitions do not have complex axioms. For example, restrictions included in classes just specify the concrete range involved in certain property (*i-properties*). In this way, queries with solution are reduced to paths of three elements. In the GALEN ontology, complex axioms are included in class definitions, with numerous anonymous classes and properties. Thus, richer queries can be realized, obtaining very specific and useful fragments. For example, the first query in the GALEN ontology allow users to know which counting

Table 2. Query examples and fragment features for the NCI ontology

nciThesaurus.owl					
OntoPath Query	**Size (Kb)**	**Classes**	**Properties**	**i-properties**	**Red.**
Whole Ontology	32850	27652	109	13961	100%
Anatomy Kina	413	2204	0	0	1.2%
*/?/Cell	4581	16969	16	444	13.3%
FindingsAndDisordersKind/?/*	3403	10672	6	2556	9.9%
OrganismKind /?/FindingsAndDisordersKind	578	2217	1	0	1.7%
GeneKind/?/FindingsAndDisorders Kina	2977	9345	2	844	8.6%
/rDiseaseHasAssociatedAnatomy/	1509	4826	1	630	4.4%
Cell/?/*	447	2204	2	274	1.3%
Tissue/?/*	425	2204	2	92	1.2%
*/?/Tissue	4563	16969	16	290	13.2%
Protein Kind/?/*	5333	14851	13	11006	15.5%
*/?/Protein Kina	1458	5420	8	1712	4.2%
GeneKind/?/*	4593	13143	6	8392	13.3%
OrganismKind/?/*	3225	10331	3	0	9.4%
*/?/OrganismKind	1893	7544	5	2152	5.5%

[14] EVS-NCI (current version: 07.04e): ftp://ftp1.nci.nih.gov/pub/cacore/EVS/NCI_Thesaurus/

methods are associated to blood cells, and the last one allow users to know resistant sensitivity cases produced by proteins.

Finally, Table 4 shows some examples of *OntoPath* that also retrieves instances. As GALEN and NCI do not provide instances, we took instead the *wine*[15] ontology, which illustrates several constructors for individuals and nominal entities.

Table 3. Query examples and fragment features for the GALEN ontology

Galen.owl				
OntoPath Query	**Classes**	**Created[16]**	**Properties**	**i-properties**
Whole Ontology	3002	221	413	2168
AbsoluteMeasurement/?/Cell/?/LiquidBlood	29	10	2	18
*/hasState/resistant	8	0	1	2
*/Attribute/resistant	8	0	1	2
[hasSubprocess/][isFunctionOf/*]	3	1	2	2
/isFunctionOf/	86	20	2	24
/hasSubprocess/	26	5	1	11
Sensitivity[hasState/resistant][Attribute/presence/?/Protein]	10	2	3	3
CardiacPathology/?/Ischaemia/?/Chronicity/?/acute	7	2	3	5
/isStructuralComponentOf//?/Extremity	34	0	2	23
*/isSolidDivisionOf/UpperExtremity	7	0	1	6

Table 4. Query examples for retrieval of instances in the wine ontology

Wine.owl				
OntoPath Query	**Classes**	**Properties**	**i-properties**	**Instances**
Whole Ontology	76	13	101	161
*/locatedIn/Region	64	1	4	42
Region/?/*	1	2	0	36 (regions)
Wine	63	0	0	53 (wines)
Wine/?/*	72	9	61	160
Wine/?/Region	64	1	25	89
/hasSugar/	64	1	24	56
[hasSugar/][hasBody/*]	65	2	31	59

8 Conclusions

Ontology modularization [1] and segmentation [2] have gained an important weight in domains such as biomedicine, where available ontologies are huge. In these cases, several issues force the final users to work with a subset of the ontology: scalability problems in the reasoning over the whole ontology, visualization in ontology editors, partial knowledge of the domain, maintenance and extension, and use in concrete applications (i.e.: information extraction guided by ontologies).

The general aim of this work is to extract consistent, closed, and useful ontology fragments, suitable for concrete applications or for knowledge exploration purposes. In contrast with ontology modularization approaches, we do not advocate automatic

[15] Wine Ontology: http://protege.cim3.net/file/pub/ontologies/wine/wine.owl
[16] Classes created by means of pattern 4.

(formal or semi-formal) techniques to partition ontologies without the participation of the final user of the ontology, which is an important limitation because the generated modules may not be useful for concrete applications. Our proposed work follows a similar approach to [13, 2], with respect to the guided module/fragment definition, where final users define their custom knowledge, instead to work with a previously module or with the whole ontology.

Currently, we are working in the definition of a well founded framework for our fragment extraction mechanism, in order to maintain and to express formally (we mean as formally a kind o representation that a reasoning system can understand; currently the maintained references does not follows any formalism, only syntactic references) the connections with the original ontology and between fragments. These connections will allow the final user to expand its previously defined fragment with more knowledge from the original ontology (or other fragments). From the literature we can emphasize, as a good staring point, the *Safe Ontology Modularization* [10] and the $\mathcal{E}$*-Connections* [17] approaches. The former one attempts to define *safe* ontology modules, whereas the latter aims at extending the OWL syntax and semantics to represent the connection between ontology modules.

Acknowledgements

This work has been partially funded by the PhD Fellowship Program of the Generalitat Valenciana, the CICYT Project TIN2005-09098-C05-04 from the Spanish Ministry of Education and Science, and the Health-e-Child European Project.

References

1. Wang, Y., Haase, P., Bao, J.: A Survey of Formalisms for Modular Ontologies. In: IJCAI 2007. Workshop SWeCKa, Hyderabad, India (January 2007)
2. Seidenberg, J., Rector, A.: Web ontology segmentation: Analysis, classification and use. In: WWW. Proceedings of the World Wide Web Conference, Edinburgh (June 2006)
3. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: COWL: Contextualizing Ontologies. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 164–179. Springer, Heidelberg (2003)
4. Seaborne, A., Prud'hommeaux, E.: SparQL Query Language for RDF (February 2005) http://www.w3.org/TR/rdf-sparql-query/
5. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In: Proceedings WWW 2002, Hawaii, USA, USA (2002)
6. Jimenez-Ruiz, E., et al.: The Management and Integration of Biomedical Knowledge: Application in the Health-e-Child Project. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006. LNCS, vol. 4278, Springer, Heidelberg (2006)
7. Wang, L., et al.: Biostar models of clinical and genomic data for biomedical data warehouse design. Int. Journal of Bioinformatics Research and Applications (2005)
8. Paraire, J., Berlanga, R., Llidó, D.M.: Resolution of Semantic Queries on a Set of Web Services. In: Andersen, K.V., Debenham, J., Wagner, R. (eds.) DEXA 2005. LNCS, vol. 3588, pp. 385–394. Springer, Heidelberg (2005)

9. Borgida, A., Serafini, L.: Distributed description logics: Directed domain correspondences in federated information sources. In: CoopIS/DOA/ODBASE, pp. 36–53 (2002)
10. Cuenca-Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Extracting Modules from Ontologies: A Logic-based Approach. In: OWLED 2007. Proc. of the Third International OWL Experiences and Directions Workshop (2007)
11. Bao, J., Caragea, D., Honavar, V.: Towards collaborative environments for ontology construction and sharing. In: CTS 2006. International Symposium on Collaborative Technologies and Systems, pp. 99–108. IEEE Computer Society Press, Los Alamitos (2006)
12. Stuckenschmidt, H., Klein, M.: Structure-based partitioning of large concept hierarchies. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, Springer, Heidelberg (2004)
13. Noy, N., Musen, M.A.: Specifying ontology views by traversal. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 713–725. Springer, Heidelberg (2004)
14. Bhatt, M., et al.: Semantic completeness in sub-ontology extraction using distributed methods. In: Laganà, A., Gavrilova, M., Kumar, V., Mun, Y., Tan, C.J.K., Gervasi, O. (eds.) ICCSA 2004. LNCS, vol. 3045, pp. 508–517. Springer, Heidelberg (2004)
15. Volz, R., Oberle, D., Studer, R.: Implementing Views for Light-weight Web Ontologies. IEEE Database Engineering and Application Symposium (2003)
16. Jiménez-Ruiz, E., Berlanga, R.: A View-based Methodology for Collaborative Ontology Engineering: an Approach for Complex Applications (VIMethCOE). In: STICA. 1st International Workshop on Semantic Technologies in Collaborative Applications (June 2006)
17. Cuenca-Grau, B., et al.: Automatic Partitioning of OWL Ontologies Using E-Connections. In: DL 2005. International Workshop on Description Logics (2005)
18. Jiménez, E., Berlanga, R., Sanz, I., Aramburu, M.J., Danger, R.: OntoPathView: A Simple View Definition Language for the Collaborative Development of Ontologies. In: López, B., et al. (eds.) Artificial Intelligence Research and Development, IOS Press, Amsterdam (2005)
19. Jiménez-Ruiz, E., Nebot, V., Berlanga, R., Sanz, I., Rios, A.: A Protégé Plug-in-Based System to Manage and Query Large Domain Ontologies. In: 10th Intl. Protégé Conference, Budapest, Hungary (2007), http://protege.stanford.edu/conference/2007/schedule.html
20. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
21. Christophides, V., Plexousakis, D., Scholl, M., Tourtounis, S.: Optimizing Taxonomic Semantic Web Queries Using Labeling Schemes. Journal of Web Semantics 1(2) (2004)
22. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases. In: SIGMOD 1989, ACM Press, New York (1989)
23. Magkanaraki, A., Tannen, V., Christophides, V., Plexousakis, D.: Viewing the Semantic Web through RVL Lenses. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, Springer, Heidelberg (2003)
24. Means, W.S., Harold, E.R.: XML in a Nutshell A Desktop Quick Reference, January 2001, Chapter 9: XPath: http://www.oreilly.com/catalog/xmlnut/chapter/ch09.html
25. Freund, J., et al.: Health-e-Child: An Integrated Biomedical Platform for Grid-Based Pediatrics. In: Health-Grid Conference, Valencia (2006)

Taxonomy Construction Using Compound Similarity Measure

Mahmood Neshati[1] and Leila Sharif Hassanabadi[2]

[1] Web Intelligence Laboratory, Computer Engineering Department
Sharif University of Technology, Iran
neshati@ce.sharif.edu
[2] Computer Science Department, Shahid Beheshti University, Iran
l_sharif@sbu.ac.ir

Abstract. Taxonomy learning is one of the major steps in ontology learning process. Manual construction of taxonomies is a time-consuming and cumbersome task. Recently many researchers have focused on automatic taxonomy learning, but still quality of generated taxonomies is not satisfactory. In this paper we have proposed a new compound similarity measure. This measure is based on both knowledge poor and knowledge rich approaches to find word similarity. We also used Neural Network model for combination of several similarity methods. We have compared our method with simple syntactic similarity measure. Our measure considerably improves the precision and recall of automatic generated taxonomies.

1 Introduction

Semantic Web is proposed to produce information structures which are able to be processed by Automatic Agents. The well-defined information of these structures is the foundation of Machine Processing. Development of Semantic Web is totally dependent on development of these structures. Ontology is the most important structure of Semantic Web. Automatic and Semi-Automatic production of ontology can play a prominent part in development of Semantic Web.

The preliminary phase of building an ontology is Taxonomy Extraction from a domain. A high quality and precision of the produced ontology can be achieved by producing a high quality and precise taxonomy. Furthermore, conceptual taxonomies have many applications in other domains. As an example,[1] used conceptual taxonomy in text Clustering. word sense disambiguation [2] and Named Entity recognition [3] are two other well-known applications of taxonomies. Here is the formal definition of a taxonomy.[4]

Taxonomy is triplet $T = (C, root, \leq_c)$ where: C is a collection of concepts (cluster of words). For simplicity each concept can be named using the words it contains within. *root* node which shows the top element and $\leq_c$ is a partial order relation on $C \cup \{root\}$ where $\forall c \in C : c \leq_c root$.

Various methods have been used to automatically build taxonomy from a conceptual domain. Unfortunately, these methods do not have the required precision to build a practically usable taxonomy. In this paper we propose a method

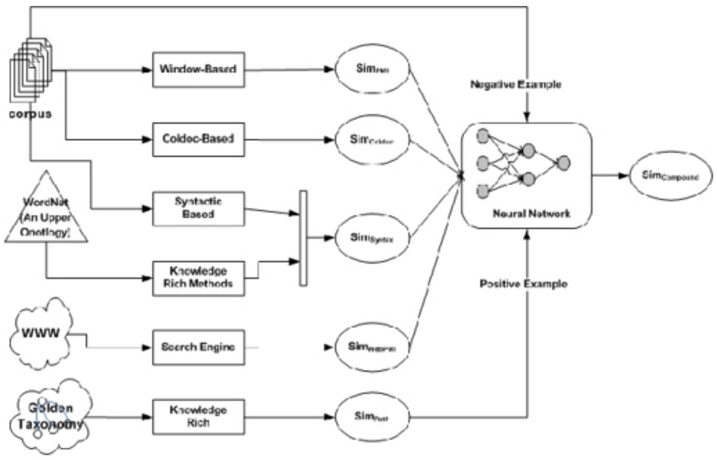


Fig. 1. Compound Similarity Measure

which increases precision of produced taxonomy. Despite other methods, we use more than one resource for extraction of taxonomic relations. Hierarchical clustering is a common method of taxonomic relation extraction. This method uses a similarity measure in order to merge most similar clusters in each step. Furthermore, Formal Concept Analysis method and set theoretical methods are applied in Taxonomy Learning [4].

Time complexity of set theoretical methods is exponential while hierarchical clustering methods have time complexity near power 2 of total number of words used in clustering, Therefore, we use hierarchical clustering method and similarity based method to build taxonomy.

Performance of hierarchical clustering method is highly dependent on similarity measure it uses, So in this paper we are going to propose a very precise and optimized similarity measure to be used in taxonomic relation extraction.

Many different methods are applied to find semantic similarity of words ([5],[6],[7]). Unsupervised methods are commonly based on Distributional Hypothesis [8]. Based on this hypothesis, different methods focus on different attributes of words to find their semantic similarity. Words with more similar attributes are considered to be similar. Co-occurrence [7], syntactic attributes [6] and occurrence in web[9] are common attributes used to find similarity of words.

We can not find a precise similarity measure by using only one of existing methods. for example Co-occurrence of words in corpus or web can be accidental sometimes and at the other hand extraction of syntactic attributes is usually faced with Data Sparseness problem[4]. So a new compound similarity measure that more precisely finds taxonomic relations seems necessary. We have used neural networks to build the compound similarity measure. The overall process is shown in fig 1.

Our contributions in this paper are:

1. We used Knowledge rich methods in order to increase performance of syntactic similarity measures while faced with Data Sparseness. Importance of syntactic attributes of words influences the syntactic similarity between them. We proposed a new weighted ranking for syntactic attributes which increases the precision of computed syntactic similarity.
2. We used neural networks to produce a compound similarity measure of words. Section 2 describes methods used to extract semantic relations of words. Section 3 describes use of neural networks to combine these extracted semantic relations. In Section 4 we'll evaluate the proposed similarity measure. Section 5 is about related works and we'll finally have a conclusion in section 6.

2 Extracting Semantic Relations

In this section we introduce methods used to extract semantic relations of words. These methods can be put in two main groups. The former methods try to find semantic relations using existing relationship structures like WordNet Ontology. These methods are called Knowledge rich because they assume existing relationship structures between words [6]. As mentioned before, we use these methods to overcome the problem of data sparseness. The latter methods called knowledge poor which do not have any assumption about semantic structure of words. Sections 2.1 and 2.2 describe these two types of methods more in detail.

2.1 Knowledge Rich Methods

WordNet[1] Ontology is the main source of knowledge rich methods to extract semantic relations. Nouns and verbs are hierarchically organized in this upper Ontology. Co-occurrence of nouns with similar verbs is a good semantic similarity measure for them as will be mentioned in section 2.2.

We use knowledge rich methods to find subjective and objective similarity of nouns. Previous methods used exact match to find subjective and objective similarity measure of nouns. In other words, Co-occurrence with mere the same verb is a factor of similarity. Despite of these methods, we assume two nouns are similar if they have subjective or objective Co-occurrence with similar verbs. We used WordNet::Similarity [10] tool to find similarity of two verbs.

PATH measure, used in our experiments, is one of the simplest methods to find similarity measure of two verbs. Shortest distance of two words represents the semantic distance of the two. $Sim_{PATH} = -\log(\frac{length(w_1,w_2)}{2D})$ This measure was proposed by [11]. In this formula $length(w_1, w_2)$ is the shortest path between w_1 and w_2 and D is the maximum depth of taxonomy. For example in fig 2 $Sim_{PATH}(w_1, w_2) = 0.12$ and $Sim_{PATH}(w_1, w_3) = 0$.

[1] http://wordnet.princeton.edu/

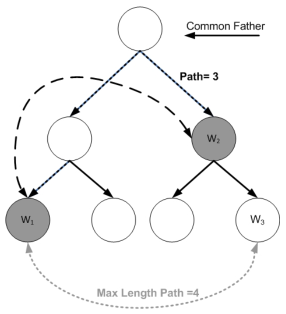


Fig. 2. PATH Similarity in taxonomy

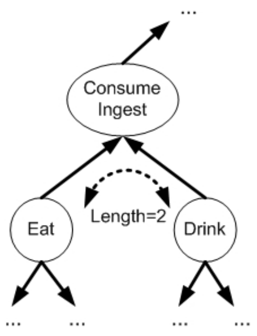


Fig. 3. Position of Eat and Drink in WordNet

The main drawback of this method is that this method assumes that all the edges of WordNet tree have the same meaning weight, But it is not true. For example length(tourist,traveler)=1 and length(tourist,person)=1 though it is obvious that tourist and traveler are more similar that tourist and person.

We used WordNet, as mentioned before, to find the similar verbs. As an example look at fig 3 that shows verbs eat and drink with distance 2 in taxonomy tree. We have two sentences extracted from tourism corpus[2] and their syntactic relationship shown in Table 1. In these sentences, nouns Cake and Coffee have

Table 1. Syntax Relation

Sentence	Extracted Relation
Its a delight stop to drink a coffee served from the back of a van.	(DOBJ[3], drink, coffee)
The Mid-Autumn Festival is the time to eat tasty moon cakes.	(DOBJ, eat, cake)

[2] We used Tourism corpus in learning phase.

Co-occurrence with verbs Eat and Drink in an object-verb relationship. We can say Cake and Coffee are similar nouns because they have similar syntactic relation with two similar verbs (According to WordNet::Similar), so these two nouns can exist in the same cluster of words. This new method uses background knowledge to extract semantic similarity of words and can overcome the problem of data sparseness.

2.2 Knowledge Poor Methods

Text Corpus and Web are two main sources for knowledge poor methods to extract semantic similarity of words.

Corpus Based Methods. We used three Text Corpus features to find semantic similarity of words. Co-occurrence in windows with predefined length, Co-occurrence in windows with length equal to the length of documents (Co-occurrence in documents) and syntactic relations are these three features used.

1. Window Based Methods

Co-occurrence of words in the same window is considered as semantic similarity of words. We assume two words are more similar if they co-occur in more common windows. We can show co-occurrence of two words using Contingency Table shown in Table 2. Each dimension represents a random discrete variable W_i with range (presence or absence of word i in a given text window). Each cell in the table represent the joint frequency of co-occurrence. Using Table2, We'll have frequency of occurrence of each word in text documents. Assume Corpus size to be n words, and consider t for the length of window, there'll be $W = n + t - 1$ windows. where W is the number of all windows.

Theorem 1. Probability of occurrence of word w_i in a window is equal to $P(w_i) = f(w_i)/W$.

Prof. We should anticipate P to maximize the Probability of occurrence of samples using Maximum Likelihood Principle. Likelihood function is defined as $L = f(X_1, X_2, ..., X_W)$ where X_i is a random sample. If the word has not occurred in i-th window then $X_i = 0$, otherwise $X_i = 1$. Assuming the independence of $X_1...X_W$ we can write: $L = f(X_1)...f(X_W)$. Probability of occurrence of w_i in n-th instance has Bernoli distribution with parameter P so we'll have:

$$L = P^{x_1}(1-p)^{1-x_1}...P^{x_W}(1-p)^{1-x_W} = P^{\sum x_i}(1-P)^{W-\sum x_i} \tag{1}$$

Table 2. Contingency Table

	w_i	$\neg w_i$	
w_j	f_{w_i,w_j}	$f_{\neg w_i,w_j}$	f_{w_j}
$\neg w_j$	$f_{w_i,\neg w_j}$	$f_{\neg w_i,\neg w_j}$	$f_{\neg w_j}$
	f_{w_i}	$f_{\neg w_i}$	

Applying logarithms in both sides of equation 1 we'll get to:

$$\ln(L) = (\sum(x_i)\ln P + (W - \sum(x_i))\ln(1-P)) \tag{2}$$

To have L maximized we should have $\ln L$ maximized so $\frac{\partial \ln L}{\partial P} = 0$:

$$\frac{\partial((\sum(x_i)\ln P + (W - \sum(x_i)\ln(1-P))))}{\partial P} = 0 \tag{3}$$

from equation 3 we have $P = \frac{\sum x_i}{W} = \frac{f(w_i)}{W}$. $\sum x_i$ is the number of windows where W_i has occurred.

Theorem 2. Probability of co-occurrence of words w_i and w_j in the same window is $P(W_i, W_j) = \frac{f(W_i,W_j)}{W}$.

Prof. Prof is like Theorem 1.
We can show co-occurrence of two words with random variable $Z = (W_i, W_j)$ which means w_i and w_j are occurred in the same window, So

$$P(Z = (Wi, Wj)) = f(w_i, w_j)/W$$

If the two words are semantically different so their occurrences in windows are independent. Hence we'll have

$$P'(Z = (W_i, W_j)) = P(W_i)P(W_j)$$

If a considerable difference exists between values of P and P' so words W_i and W_j are semantically similar, otherwise occurrences of W_i and W_j in windows are independent and so W_i and W_j are semantically different. For measuring semantic similarity of words we should compute the difference distribution of P and P'. For this purpose, we use PMI[4]. This method was proposed by [12]. PMI similarity measure is calculated using this formula

$$Sim_{PMI}(w_1, w_2) = \log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

PMI method calculates occurrence independency of w_i and w_j. Positive values of PMI show that w_i and w_j have more co-occurrences than independent words. Negative values show that w_j is not willing to occur where w_i is occurred. Near zero values show that w_i and w_j are independent words. The more w_i and w_j are independent, the more $P(w_i, w_j)/P(w_i)P(w_j)$ inclines to 1 and PMI inclines to zero.

PMI similarity measure of some words are calculated and shown in Table 3. As told before, City and Visa are almost independent words because of their near zero PMI. (April ,October) and (City,Place) are semantically similar words because they have positive PMI.

[4] Point wise Mutual Information.

Table 3. PMI Similarity

w_i	w_j	$Sim_{PMI}(w_i, w_j)$
April	October	6.47
City	Place	6.22
City	Visa	0.38
Day	West	-0.83

2. Document Co-occurrence Similarity

Document co-occurrence similarity is some type of window co-occurrence similarity where window size is document length. We assume that words which occur in the same document are similar, so we can calculate similarity as

$$Sim_{COLDOC} = \frac{2df(w_1, w_2)}{df(w_1) + df(w_2)}$$

Where $df(w_1)$ is number of documents, w_1 occurs, $df(w2)$ is number of documents, w_2 occurs and $df(w1, w2)$ is number of documents w_1 and w_2 both occur. This similarity measure is based on Dice method. We can calculate similarity measure based on Jaccard or Overlap methods but in our experiments Dice method achieved more performance than others. Pairs of words in tourism corpus with most document co-occurrence similarity are shown in Table 4. This method has some weaknesses compared with window method. For example this method is dependent on the size of documents.Each paragraph of the document can be about a different subject, so Co-occurrence of words in big documents does not necessary mean a semantic similarity. We can normalize the size of documents to decrease the impacts of this problem but if we have big size documents the overall precision will be low.

Window Co-occurrence and Document Co-occurrence methods are commonly used to extract synonym words in literature [7]. As shown in Table 4 two words with high Document Co-occurrence measures are not necessarily synonyms but semantic relation of these pairs are undeniable. This semantic relation can be used in clustering algorithm. In section 3 we will explore the impact of each similarity measure on overall clustering process and we'll produce a weighted ranking of these different similarity measures.

Table 4. COLDOC Similarity

w_i	w_j	$Sim_{COLDOC}(w_i, w_j)$
April	October	0.65
Attraction	Museum	0.49
August	June	0.70
City	Place	0.65

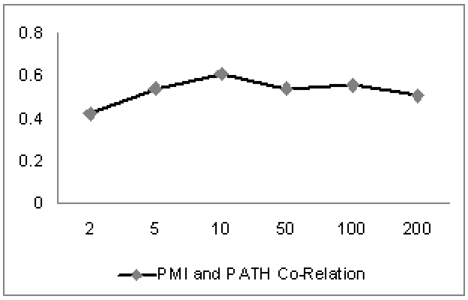


Fig. 4. Co-relation of PMI and PATH

As mentioned before two words can occur in the same document but in different paragraphs, so they can have no similarity. We combine Window Co-occurrence and Document Co-occurrence methods to overcome this problem. In other words, we consider two words similar if they occur in the same document and have a distance less than t, the size of window. We computed co-relation coefficient of PMI and PATH for 100 pairs of tourism domain words existing in WordNet ontology in order to find an appropriate t for window size.(fig 4) Before, we normalize similarity measures between 0 and 1. The most Co-relation coefficient is achieved when the size of windows is 10. Although we have maximized the co-relation coefficient but they are still low. This shows that window method is not solely adequate to compute similarity measures in taxonomy learning methods.

3. Syntactic Co-occurrence

We can use syntactic Co-occurrence of words as another poor knowledge method for similarity computation. The main idea here is that words which co-occur with similar Noun or Verbs in the same syntactic roles are possibly semantically similar.

Just like window and document co-occurrence methods this method is based on Distributional Hypothesis[8].We can define a signature for each word. As shown in Table 5 Iran, Cyprus and Country have an Object relation with the same verb Visit. We may be able to find a semantic similarity using syntactic relationships. Syntactic method needs linguistic information. type of words should be extracted using a POS Tagger. Noun and Verb phrases can be extracted using a

Table 5. Syntactic similarity

Sentence	Extracted Relation
Everyone needs a visa to visit Iran.	(Iran, DOBJ, Visit)
April, May, September and October are the most pleasant times, climatically, to visit Cyprus.	(Cyprus, DOBJ, Visit)
The best time to visit the country is in the late spring.	(Country, DOBJ, Visit)

tagged text corpus. Then signature of each word is calculated using syntactically analyzed text corpus. Stanford Parser and tagger[5] is used as a POS Tagger in our experiments. Our text corpus is extracted from Web[6]. This corpus contains 1801 text files. There are syntactic errors in some sentences and furthermore output of parser is wrong sometimes, so we should analyze quality of the extracted relations using statistical techniques.

We can show each extracted relation with triplet (w_1, r, w_2) where r is type of relation and w_1 and w_2 are words participating in relation r. We show collection of triplets as T, text corpus words as W and syntactic roles as Rel. Having these triplets we can build signatures for each word. Each word signature has its role and a word which has syntactically co-occurrence with it which constitute a triplet.

$$Signature(w) = \{(r, w')|(w, r, w') \in T, r \in Rel, w' \in W\}$$

For each relation r in Rel we can consider a collection of w's signatures which contain words $w_1 ... w_n$ relating to w with r. In other words $Signature_r(w)$

$$Signature_r(w) = \{(w')|(w, r, w') \in T, w' \in W\}$$

contains all the words having relation r with w.

Among these words having relation r with w exist words which are accidentally occurred within corpus relations and some of them can be errors of Parser program, so we should analyze the triplets and select non-accidental triplets which have more frequency of occurrence than accidental ones. Furthermore we should anticipate the importance of a triplet with a numerical measure. We can model occurrence of triplet (w, r, w') within text corpus by using co-occurrence of three event.[5]

- W: a random selected word is w
- R: a random selected role is r
- W': a random selected word is w'

Probability of this co-occurrence is calculated by:

$$P(W, R, W') = P(R)P(W|R)P(W'|W \cap R)$$

Assume that occurrence of W and W' is independent so $P(W'|W \cap R) = P(W'|R)$ and we we'll have :$P(W'|W \cap R) = P(W'|R)$ $P(W, R, W')$ can be find by:

$$P(R) = \frac{N(r)}{N(Rel)}$$

$$P(W|R) = \frac{P(W \cap R)}{P(R)} = \frac{N(Signature_r(w))}{N(r)}$$

[5] www-nlp.stanford.edu/downloads/lex-parser.shtml
[6] http://www.lonelyplanet.com

$$P(W'|R) = \frac{P(W' \cap R)}{P(R)} = \frac{N(Signature_r(w'))}{N(r)}$$

finally we have:

$$P(w, r, w') = \frac{N(Signature_r(w))N(Signature_r(w'))}{N(r)N(Rel)}$$

and Information Content will be

$$IC_1(P(w, r, w')) = -\log$$

$$\frac{N(Signature_r(w))N(Signature_r(w'))}{N(r)N(Rel)}$$

on the other hand by using Maximum Likelihood Principle we can find probability of occurrence of (w, r, w') in corpus by:

$$P_{MLE}(w, r, w') = \frac{N(w, r, w')}{N(Rel)}$$

Where its Information Content is

$$IC_2(P(w, r, w')) = -\log \frac{N(w, r, w')}{N(Rel)}$$

The difference of IC2 and IC1 can be a measure of quality for each triplet. In other words we choose triplets which have an Information Gain more than a threshold t. So we'll have a new definition for signatures:

$$Signature(w) = \{(r, w')|(w, r, w') \in T, r \in Rel,$$

$$w' \in W \, IG(w, r, w') > t\}$$

Assuming the above formula Information Gain will be

$$IG(W, r, W') = \log \frac{N(w, r, w')N(r)}{N(Signature_r(w))N(Signature_r(w'))}$$

A triplet's Information Gain shows how much reliable that triplet is. In other words triplets with a low frequency of occurrence have low Information Gain. Furthermore; Information Gain of a triplet shows how much semantically valuable its syntactic relation is. As an example considers Table 6, Information

Table 6. Information content of syntactic relations

Triplet	$IG(w, r, w')$
(Visit, DOBJ, Iran)	4.96
(Visit, DOBJ, Area)	1.3
(Visit, DOBJ, Day)	-0.1

Table 7. Signature of Iran and Germany Extracted from text

Relatoin	Iran(word,IG)	Germany(word,IG)
AMOD	(northern,8.18)	(southern, 8.08)
APPOS	(west, 10.1)	(republic, 10.01)-(connection, 9.01)
DOBJ	(visit,5.59)-(adjoin, 11.36)-(face, 7.55)	(visit, 4.01)-(abut, 9.77)
NSUBJ	(experience,7.79)	(play, 9.58)
NSUBJPASS	(bless, 7.73)	-
POSS	(town,5.53)-(mountain,6.86)-(holiday, 6.45)	(river, 6.45)-(treasure, 7.18)

Content of Iran with verb Visit is 4 times of Information Content of Area with visit and the syntactic relation is an Object-Verb one. Area participates with 40 nouns in an Object-Verb relationship; Iran participates only in 2 relationships as an object so Information Gain of (Iran , Visit , Object-Verb) is more than Information Gain of (Area, Visit, Object-Verb). Information Gain of (Day, Visit, Object-Verb) is negative. It can be explained as a POS Tagger error in recognizing the syntactic relation, so this triplet will be neglected. It can be said that Information Gain in co-occurrence of a verb with a general noun is less than Information Gain in co-occurrence of the same verb with a more specific noun and also Information Gain in co-occurrence of a noun with a general verb is less than Information Gain in co-occurrence of the same noun with a more specific verb. Signatures of Iran and Germany are shown in Table 7 Using this method to find signatures of words, we encounter the problem of Data Sparseness. For example Visit is the only verb with which Iran and Germany both have an Object-Verb relation. Although abut and adjoin are different verbs; they are semantically similar. A very big text corpus (web as an example) can be used to solve the problem of Data Sparseness. We used WordNet to overcome this problem instead. For example in DOBJ relation we can use WordNet Ontology to extract semantic similarity of abut and adjoin.

Information Gain of each binomial is a measure for semantic value of its occurrences in corpus, So co-occurrence with a frequent signature has less Information Gain that co-occurrence with a more specific (less frequent) signature.[4] Used solely shared signatures of words to calculate semantic similarity of words, we; furthermore; considered Information Gain of words to calculate semantic similarity of words. In order to calculate semantic similarity measure of W_1 and W_2; for each W' having relation r with W_1 we find the most similar word to W' among words which have relation r with W_2 and name its similarity as $MaxSim(W')$ so

$$Sim_{w_1 \to w_2}(w_1, w_2) =$$
$$\frac{\sum_{r \in R}(\sum_{(w',r) \in signature'(w)} Max_{sim}(w') * IG(w, r, w'))}{\sum_{(w',r) \in signature'(w)} IG(w, r, w')}$$

Table 8. Co occurrence in web

w_1	w_2	Sim_{Web_PMI}
elephant	proboscidean	7.15
sun	star	6.82
animal	apple	0.54
year	time period	-4.5

The value of the similarity above will be between 0 and 1. This measure is a directed measure so we use below formula to find a symmetric measure.

$$Sim_{Syntax}(w_1, w_2) =$$
$$\frac{Sim_{w_1 \rightarrow w_2}(w_1, w_2) + Sim_{w_2 \rightarrow w_1}(w_1, w_2)}{2}$$

Web based methods. Text corpus based methods usually encounter with the problem of Data Sparseness. Using web as a big text corpus can solve this problem. Web has multi billion WebPages. This size of data can not be processed directly, so we need an interface to extract the required data in a reasonable time period. Search Engines as in Fig1 can be used as an interface to extract required information from web. Number of retrieved pages can be a measure of semantic availability of our query. As an example if we query (tourist , city) and (tourist , computer) in Search Engine as expected number of retrieve pages for the former is 4 times of the latter, So if we query the words of a cluster, number of retrieved pages should show a semantic availability.

If we search the web with similar semantic words, Number of retrieved pages will considerably be more than Estimate of independent occurrence of them. In other words if $P(w_1)$ and $P(w_2)$ are probability of occurrence of w_1 and w_2 in web and assume the occurrence of w_1 and w_2 is independent of each other, Then $P(w_2|w_1) = P(w_2)$ so $E(w_1, w_2) = P(w_1)P(w_2) * N(web)$. Comparing $E(w_1, w_2)$ and $N(w_1, w_2)$ yields to a measure for semantic relation of w_1 and w_2. This measure is like PMI mentioned in 2.2.

$$Sim_{Web\_PMI} = \log \frac{\frac{N(w_1 \cap w_2)}{N_{WEB}}}{\frac{N(w_1)N(w_2)}{N_{WEB}N_{WEB}}}$$

Sim_{Web_PMI} inclines to zero when occurrences of w_1 and w_2 are independent. If w_1 and w_2 share a similar occurrence pattern Sim_{Web_PMI} will be a positive number. N_{WEB} is number of pages indexed by search Engine[7]. N_{WEB} is near$10^1 0$ now. Some co-occurrence examples are shown in Table 8.

3 Comparison and Combination of Measures

We introduced a few similarity measures of words, but we don't know how much practically usable each measure is and how much each measure can impact the

[7] We used Google as search engine.

construction of a hierarchical taxonomy based on semantic similarity of words. Section 3.1 describes how we produced a compound measure of semantic similarity using a neural network model.

3.1 Combination Algorithm

We produce a vector for each pair of words which contains calculated measures mentioned in previous sections as its elements. Vector extraction algorithm is shown below:

```
**Algorithm 1- Generate_Vector**($w_1, w_2$)
// this Algorithm generate a vector for each pair($w_1, w_2$), $C$ is a Corpus
$Sim_{PMI}(w_1, w_2)$ ←Window-Based($w_1, w_2, C$)
$Sim_{Coldoc}(w_1, w_2)$ ←Document-Colocation($w_1, w_2, C$)
$Sim_{Syntax}(w_1, w_2)$ ←Syntax-Based($w_1, w_2, C, WordNet$)
$Sim_{Web\_PMI}(w_1, w_2)$ ←Web-Based($w_1, w_2, Web$)
Vector $v = (Sim_{PMI}, Sim_{Coldoc}, Sim_{Syntax}, Sim_{Web\_PMI})$
Normalize($v$)
Return $v$
```

We should find impact of each measure on hierarchical taxonomy to build a weighted ranking of these measures. In other words if we know Path Similarity of two words, we can have an estimation for impact of each measure to get to the Path Similarity. In order to find impacts (weights) we used a neural network model and Sensitivity Analysis. Using PATH method we calculated the goal variables.

We used a Tourism Taxonomy as our reference which contains 236 words. These words are leaves of taxonomy tree. Not leaf nodes are abstract concepts which may not have an assigned name in text corpus. Our goal is to produce a very similar taxonomy from text corpus automatically, so we need a compound similarity measure which produces that taxonomy. Therefore we use the hand-built taxonomy in learning phase of neural network to find the compound measure.

Algorithm 1 produces a 4-dimensional vector for each pair of words. In test phase we used another hand-built taxonomy for finance domain. We omitted words which exist in taxonomy but don't exist in text corpus. For each pair of words using Algorithm 1 we produced the vector. The weighted ranking which is the output of neural network learning phase is applied on each vector to produce a compound similarity measure. Finally we built our taxonomy using Hierarchical Clustering Algorithm. The overall production of compound similarity measure is shown in [1].

4 Evaluation

We need a method to find how much the automatically built taxonomy is similar to a real taxonomy in order to analyze our compound similarity measure.[13]

used true "IS-A" relations to find how real the built taxonomy is. Our algorithm just builds the clusters and can not assign a name to each cluster so we can not use this method, So to analyze our compound similarity measure we compare our automatically built taxonomy with a hand-built one. We used Tourism taxonomy to in neural network's learning phase. We rarely encounter with semantic ambiguity or synonyms in specific domains, so for analyzing the produced measure we use another specific domain (Financial Taxonomy). We use Taxonomy Overlap method introduced by [14] to find similarity of taxonomies.

We name each taxonomy concept (internal nodes) with the words it contains. Semantic Cotopy introduced by [14] is a measure to find the similarity of two taxonomies.

$$SC(c_i, T_1, T_2) = \{c_j \in C_1 \cap C_2 | (c_j leq_{c_1} c_i) \cup (c_i leq_{c_1} c_j)\}$$

Using Semantic Cotopy we can show each concept by its predecessors and ancestors concepts. So we can find similarity of two concepts by comparing collection of ancestors and predecessors and finding shared concepts of them.[14] defines Taxonomy Overlap as [4]

$$TO(T_1, T_2) = \frac{1}{|C_1 - C_2|}$$

$$\sum_{c \in C_1 - C_2} max_{c' \in c_2 \cup \{root\}} \frac{|SC(c, T_1, T_2) \cap SC(c', T_2, T_1)|}{SC(c, T_1, T_2) \cup SC(c', T_2, T_1)}$$

In other words for each automatically built concept we find the most similar concept in hand-built taxonomy. Finally we find the average Semantic Cotopy of concepts which do not exist in hand-built taxonomy by calculating Taxonomy Overlap measure shown in the above formula. Look at Fig 5.

Consider right hand taxonomy as an automatically-built taxonomy and left hand taxonomy as a hand-built reference taxonomy. We should analyze concepts which exist in automatically built taxonomy and do not exist in reference taxonomy to find out how much similar (between 0 and 1) they are with the concepts

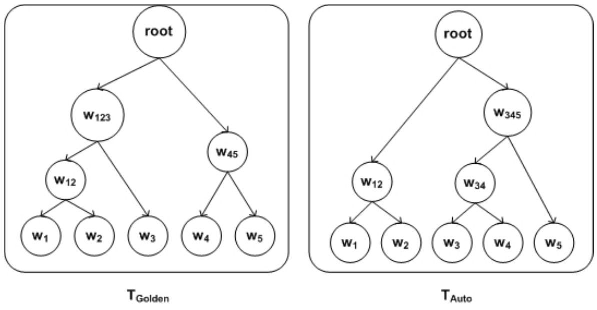


Fig. 5. Comparing Taxonomies

of reference taxonomy. Concepts W_{34} and W_{345} are the only automatically produced concepts which don't exist in reference taxonomy. By computing SC for each concept in both Taxonomies we find that the most similar concept in golden taxonomy to W_{34} and W_{345} is W_{45}. So we have

$$P(T_{Auto}) = TO(T_{Auto}, T_{Golden}) = 0.50$$

If we replace the two taxonomies in Taxonomy Overlap measure formula and calculate the Taxonomy Overlap measure we can estimate the Recall of the automatically built taxonomy. In other words how many reference concepts are produced automatically.

$$R(T_{Auto}) = TO(T_{Golden}, T_{Auto}) = 0.42$$

Hierarchical Clustering Algorithm merges two clusters if similarity of them is more than threshold t.In fact we control quality of the clusters by value of t. Semantic distance of words within a cluster decreases when we increase t so the Precision goes up. This increase in Precision may yield a decrease in Recall. We analyzed performance of the compound measure by a comparison of Precision and Recall of the taxonomy automatically built by Syntactic Similarity Method without Knowledge Rich (used by [4]) with Precision and Recall achieved in our experiment.

Precision-Recall diagram for the compound measure and syntactic measure is shown in Fig 6. The compound measure has more Recall value in acceptable Precisions than syntactic measure. We used 8 threshold values (0,0.1,...,0.8) as t for the precision-recall diagram of Fig 6. For t=0.7 syntactic measure doesn't merge any cluster and puts all the words directly under root. This happens in t=0.8 for the compound measure.

To have a better comparison we use Harmonic mean of Precision and Recall

$$F(T_{Auto}) = \frac{2P(T_{Auto})C(T_{Auto})}{P(T_{Auto} + C(T_{Auto}))}$$

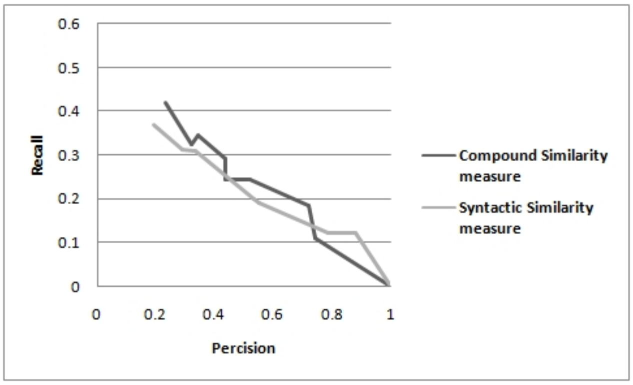


Fig. 6. Comparing compound and syntactic measures

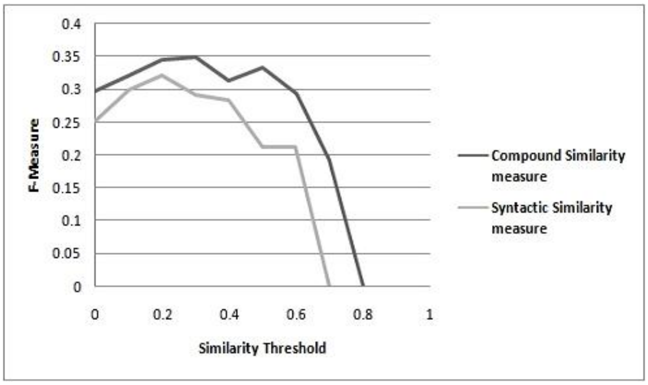


Fig. 7. Comparing F-Measures of compound and Syntactic measures

Fig 7 shows F measure value for the compound measure is higher than syntactic measure. Maximum F measure (0.32) yields when t=0.2.

As shown in diagrams 6 and 7 performance of the compound measure is considerably higher than syntactic measure. As mentioned before Text Corpus based methods generally and Syntactic methods specifically face with the problem of Data Sparseness. This data sparseness can effectively decrease F measure. We can solve this problem by our compound measure. Furthermore use of multiple semantic resources (text corpus , Web , WordNet) has increased the precision of the compound measure and subsequently the precision of the concept learning algorithm.

5 Related Work

There are many researchers working on Taxonomy learning algorithms. [4] used Formal Concept Analysis to extract taxonomy relationships. In [15] WordNet is used as a semi supervisor while merging clusters. Although this method is based on Knowledge rich approach but the way we used WordNet is completely different. [16] used multiple resources to extract taxonomy relationships but it did not use machine learning techniques to combine these resources.

Another group of taxonomy extraction methods make use of syntactic-linguistic patterns. [17] Extracts "Is-a" relations using predefined patterns. [18]Used the same idea to find "Part-of" relationships. These methods achieve a high precision but like other methods face with the problem of data sparseness and low recall.

As mentioned before we can solve the problem of Data Sparseness by using a big text corpus like web. [19]extracts Hearst patterns from web using Google search engine . [20] Uses existing resources like Wiki and a German dictionary to extract ontology. [21] Uses LSI method to increase the quality of clusters and naming them.[22] Enriches existing taxonomies by similarity and specificity measures. As told in [22] introduced algorithm can't be used to extract a complete taxonomy from scratch.[9] Makes use of word co-occurrences and Snippets of retrieved pages to extract synonyms from web.[23] Tries to find similarity of two passages using Knowledge Rich approach and WordNet. In order to review the state of the art in ontology learning refer to [24].

6 Conclusion

In this paper we introduced a compound measure of word similarity and used it to cluster and extract taxonomy from a specific domain. Despite previous methods, similarity measure introduced in this paper makes use of text resources, Knowledge Rich and search engines in order to increase precision. We used a learning collection and a neural network model in order to combine different measures. Taxonomy extracted by our compound measure has considerably more precision-recall value in comparison with the reference taxonomy extracted by syntactic measure. Our experiments show by using concurrent and combined resources we can overcome the problem of Data Sparseness.

References

1. Hotho, A., Staab, S., Stumme, G.: Ontologies improve text document clustering. In: Proceedings of the 2003 IEEE International Conference on Data Mining, pp. 541–544. IEEE Computer Society Press, Los Alamitos (2003)
2. Resnik, P.: Semantic similarity in a taxonomy: An information–based measure and its application to problems of ambiguity in natural language. Journal of Artificial Intelligence Research 11, 95–130 (1999)
3. Hahn, U., Schnattinger, K.: Towards text knowledge engineering. In: AAAI/IAAI, pp. 524–531 (1998)
4. Cimiano, P., Hotho, A., Staab, S.: Learning concept hierarchies from text corpora using formal concept analysis. Journal of Artificial Intelligence Research (JAIR) 24, 305–339 (2005)
5. Lin, D.: Automatic retrieval and clustering of similar words. In: COLING-ACL, pp. 768–774 (1998)
6. Gasperin, C., Gamallo, P., Agustini, A., Lopes, G., de Lima, V.: (Using syntactic contexts for measuring word similarity) available at: `http://citeseer.ist.psu.edu/article/gasperin01using.html`
7. Terra, E., Clarke, C.: Frequency estimates for statistical word similarity measures. In: Proceedings of Human Language Technology conference North American chapter of the Association for Computational Linguistics, pp. 244–251 (2003)
8. Harris, Z.: Mathematical Structures of Language (1968)
9. Bollegala, D., Matsuo, Y., Ishizuka, M.: Measuring semantic similarity between words using web search engines. In: WWW 2007. Proceedings of the 16th international conference on World Wide Web, pp. 757–766. ACM Press, New York (2007)
10. Pedersen, T., Patwardhan, S., Michelizzi, J.: Wordnet: Similarity - Measuring the relatedness of concepts. In: AAAI, pp. 1024–1025 (2004)
11. Leacock, C., Chodorow, M.: Combining local context and wordnet sense similiarity for word sense disambiguation. MIT Press, Cambridge (1998)

12. Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. In: Proceedings of the 27th. Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, pp. 76–83 (1989)
13. Caraballo, S.: Automatic construction of a hypernym-labeled noun hierarchy from text. In: Proceedings of the Conference of the Association for Computational Linguistics (1999)
14. Maedche, A., Staab, S.: Measuring similarity between ontologies. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 251–263. Springer, Heidelberg (2002)
15. Cimiano, P., Staab, S.: Learning concept hierarchies from text with a guided hierarchical clustering algorithm. Available at: `http://citeseer.ist.psu.edu/article/cimiano05learning.html`
16. Cimiano, P., Pivk, A., Schmidt-Thieme, L., Staab, S.: Learning taxonomic relations from heterogeneous sources. In: ECAI 2004. Proceedings of the Ontology Learning and Population Workshop (2004)
17. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Proceedings of the 14th conference on Computational linguistics, Association for Computational Linguistics, pp. 539–545 (1992)
18. Berland, M., Charniak, E.: Finding parts in very large corpora. In: Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics, Association for Computational Linguistics, pp. 57–64 (1999)
19. Cimiano, P., Staab, S.: Learning by googling. SIGKDD Explor. Newsl. 6(2), 24–33 (2004)
20. Weber, N., Buitelaar, P.: Web-based ontology learning with ISOLDE. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) ISWC 2006. LNCS, vol. 4273, Springer, Heidelberg (2006)
21. Ramakrishnan, Cartic, E.A.: TaxaMiner: Improving taxonomy label quality using latent semantic indexing (2005)
22. Ryu, P.M., Choi, K.S.: Taxonomy learning using term specificity and similarity. In: Proceedings of the 2nd Workshop on Ontology Learning and Population: Bridging the Gap between Text and Knowledge, Association for Computational Linguistics , pp.41–48 (2006)
23. Zanzotto, F.M., Moschitti, A.: Automatic learning of textual entailments with cross-pair similarities. In: ACL 2006. Proceedings of the 21st International Conference on Computational Linguistics, pp. 401–408 (2006)
24. Shamsfard, M., Barforoush, A.A.: The state of the art in ontology learning: a framework for comparison. Knowl. Eng. Rev. 18(4), 293–316 (2003)

r^3– A Foundational Ontology for Reactive Rules[⋆]

José Júlio Alferes and Ricardo Amador

Centro de Inteligência Artificial - CENTRIA, Universidade Nova de Lisboa
{jja,ra}@di.fct.unl.pt

Abstract. In this paper we present the r^3 ontology, a foundational ontology for reactive rules, aiming at coping with language heterogeneity at the rule (component) level. This (OWL-DL) ontology is at a low (structural) abstraction level thus fostering its extension. Although focusing on reactive rules (reactive derivation rules not excluded), the r^3 ontology defines a vocabulary that allows also for the definition of rule (component) languages to model other types of rules like production, integrity, or logical derivation rules.

1 Introduction

The goal of the Semantic Web is to bridge the heterogeneity of data formats and languages and provide unified view(s) of the Web. In this scenario, XML (as a format for storing and exchanging data), RDF (as an open abstract data model), OWL (as an additional logic model), and WSDL2 (as a semantically extensible service model) provide the natural underlying concepts.

The Semantic Web does not have any central structure, neither topologically nor thematically, rather it is based on peer-to-peer communication between autonomous, and autonomously developing, nodes. Furthermore, the Semantic Web should be able not only to support querying, but also to propagate knowledge and changes in a semantic way. This *evolution* and *behavior* depends on the cooperation of nodes. In the same way as the main driving force for RDF and the Semantic Web idea was the heterogeneity and incompleteness of the underlying data, the heterogeneity of concepts for expressing behavior requires an appropriate handling on the semantic level. Since the contributing nodes are prospectively based on different concepts, such as data models and languages, it is important that *frameworks* for the Semantic Web are modular, and that the *concepts* and the actual *languages* are independent. Even if we would agree that for querying the current set of "common" standards for particular data/knowledge representations/models (e.g. XQuery for XML vs. SPARQL for RDF) could evolve into a single universal query language, which is doubtful, the concepts for describing and implementing behavior are much more different, due to different needs, and it is really unlikely that there will be a unique language for the latter throughout the Web.

Heterogenous Reactivity. In this setting, *reactivity* and its formalization as *Event-Condition-Action (ECA) rules* provide a suitable common model because they provide a

⋆ This research has been funded by the European Commission within the 6th Framework Programme project REWERSE number 506779.

modularization into clean concepts with a well-defined information flow. An important advantage of them is that the *content* of a rule (event, condition, and action specifications) is separated from the *generic semantics* of the ECA rules themselves which has a well-understood meaning: when an event (atomic or composite, the latter possibly using some event algebra for composition) occurs, evaluate a condition (possibly gathering further data via queries, again possibly combined via an algebra of queries), and if the condition is satisfied then execute an action (or a sequence of actions, a program, a transaction, or even start a process). Another important advantage of ECA rules is their loosely coupled inherent nature, which allows for declaratively combining the functionality of different Web Services (providing events and executing actions). ECA rules constitute a generic uniform framework for specifying and implementing communication, local evolution, policies and strategies, and –altogether– global evolution in the Semantic Web.

Previously, in [16,17] we have proposed an ontology-based approach for describing (reactive) behavior and evolution in the Web, following the ECA paradigm. This work also defines a global architecture and general markup principles for a modular framework capable of *composing* languages for events, conditions, and actions by separating the ECA semantics from the underlying semantics of events, conditions and actions. This modularity allows for high flexibility wrt. the heterogeneity of the potential sub-languages, while exploiting and supporting their meta-level *homogeneity* on the way to the Semantic Web. The interested reader is referred to [2,1] for additional details on the present state of this work.

Semantic Web Events. The notion of event is core to ECA rules, and in a reactive model of behaviour for the Web it needs to be freed of limitations introduced by more specific settings (e.g. active databases). Events in the Web cannot be restricted to the realization of a specific set of actions (e.g. insert, update and delete). Instead a *Web Event* is to be understood as the actual perception by a reactive system of an(y) external (or otherwise uncontrolled) occurrence, that may or may not be the result of a known action.

Logically, an event may be perceived as a temporary (non-persistent) assertion, resulting in the evolution of the knowledge base, as described in [4], through concrete (re)actions (or active deductions) that may generate new persistent assertions, invalidate existing assertions or cause additional externally perceivable occurrences (i.e. events). The notion of non-persistence of an event is of utmost importance for the Web given the humongous number of events perceivable in such a global system. This global nature also precludes any solution based on undiscriminated broadcast of events; systems interested in particular kinds of events have to express their interest to specialized event brokers. The latter may to some extent persist historical event information, but reactive rule engines have to be free of such a burden.

In a distributed environment formed of autonomous nodes, like the (Semantic) Web, ECA rules can not react to actual occurrences, only (more or less reliable) perceptions of those occurrences are generally available, and even those may sometimes go unnoticed. Nevertheless, using an eclectic mix of deductive and reactive rules, and based on different lower level perceptions, one may achieve a symbolic definition of higher level events that (fully) abstract and mimic (to the extent of the knowledge they represent) the actual occurrences, possibly even compensating for unperceived ones through alternative or implicit perceptions; thus allowing to shift the focus to *Semantic Web Events* (like *a book has been bought online*), eventually abstracting away the intricacy of Web Events (just try to imagine how many different ways exist to perceive that a book has been bought online).

Resourceful Reactive Rules. Since the inception of the Semantic Web, rules have always been proposed as one of its upper layers: an ontology-based one. Although much research effort is being targeted upon defining rules for and about ontologies, pragmatical and compatibility issues seem to be guiding the work on modelling rules themselves. In what concerns the latter, most of the current proposals are based on XML markups (e.g. [7]); eventually relying on specific abstract syntax for defining rule semantics (e.g. [8,14]). Markup-based approaches, as such, seem to ignore the fact that rules do not only operate on the Semantic Web, but are themselves part of it. In general (ECA) rules and their components must be communicated between different nodes, and may themselves be subject to being queried and updated, especially if one wants to reason about evolution, leading to a Semantic Web capable of dynamic behaviour according to behaviour policies. For that, (ECA) rules themselves must be first class citizens of the Semantic Web. This need calls for a foundational ontology for describing (ECA) rules. Such an ontology, according to the heterogeneity requirement previously presented, must provide also the means to describe different languages to be used at the rule (component) level. As such, we make two important assumptions: first, in the Semantic Web, rules are resources like everything else, and secondly, there won't be such a thing as a (concrete) universal rule language (particularly in what concerns ECA rule components). Given these two hypotheses r^3 takes a third hypothesis: ontologies, in OWL-DL, provide a suitable tool for describing language heterogeneity.

Present State. Although the examples included in [16] use "syntactical" languages in XML term markup –ECA-ML– to describe ECA rule components, as stated there, *also languages using a semantical, e.g., OWL-based representation (which have to be developed) can be used*; thus leading to fully embrace the approach proposed in [17]. To further experiment with both approaches, namely syntactic and semantic, two REWERSE WGI5[1] sub-projects, aiming at developing prototypes of the proposed general ECA framework, were launched: MARS [22] and r^3 [23]. Currently, both prototypes are functional, available online, and eventually integrable through appropriate syntactic/markup transformations. The MARS project is now also evolving into the semantic level taking a flexible approach, not restricted to OWL-DL; future integration of the two prototypes is to be pursued at this semantic/ontology level. The interested reader may find additional details on both prototypes in [2,1].

In this paper, results of the r^3 project on defining an (OWL-DL) foundational ontology for reactive rules are presented. The current proposal is at a low (structural) abstraction level; the extension of this proposal towards characterizing higher abstraction level concepts, like domain/application specific languages (vs. algebraic and general-purpose languages) is not excluded, and fruitful synergies are expected with the MARS project which is following an higher level approach. Although focusing on reactive rules, the r^3 ontology defines a vocabulary allowing for the definition of rule and rule component languages to model also other types of rules.

[1] REWERSE WGI5: Evolution and Reactivity - http://rewerse.net/I5/

Related Work. To the best of our knowledge, to the present there are only three ontology proposals for describing rules: SWRL [13], WRL [5] and SBVR [18]. Loosely speaking, the rules modelled by SWRL and WRL are Horn rules; none of the two includes any form of reactive rules. SWRL provides an OWL (Full) ontology; WRL includes a mapping to OWL-DL (but only at the core level that does not include rules). Although following different approaches, both proposals "extend" OWL by providing means to express OWL-DL axioms. On the other hand, SBVR, which is not formalized in OWL terms, does not exclude reactive rules (some illustrative examples are even present in the specification); but it explicitly chooses not to address its specificities, postponing such matters for reevaluation upon OMG's BPDM results. About SBVR, it is worth mentioning, that it is targeted to describe business rules in general with an emphasis on human understanding [21] (which may hinder machine computability) and it is the only one of the three that actually addresses the issue of language heterogeneity (introducing the concept of business vocabularies, as a form of controlled natural language). Nonetheless, it must be stressed that, SBVR is the only one of these three that does not include a formalization of its semantics.

Most current standardization efforts related to rule interchange, e.g. [8,7], by following a markup-oriented approach, tend to be charged with syntactical details without semantic value, which has a negative impact on any attempt to raise them to the ontology level. Concrete syntax is usually expressed in terms of abstract syntax, not the other way around. Nevertheless, one of such efforts has to be mentioned even if it does not include any Semantic Web transparent proposal: Common Logic (CL) [14], in what concerns language heterogeneity, is probably the standardization effort closest to the spirit of $\boldsymbol{r^3}$. CL achieves semantics formalization in face of language heterogeneity by limiting its family of languages to those that (and we quote) *have declarative semantics* and *are logically comprehensive*, i.e. *it is possible to understand the meaning of expressions in these languages without appeal to an interpreter for manipulating those expressions* and, *at its most general, they provide for the expression of arbitrary first-order logical sentences*. Given the state of the art, this limitation actually excludes most forms of reactive rules from CL.

Structure of the Paper. We start (in section 2) by introducing the $\boldsymbol{r^3}$ ontology from a rule "taxonomy" point of view. In the following sections we detail the $\boldsymbol{r^3}$ ontology explaining and illustrating[2] how to define different languages (in section 3), and how to use these languages to define heterogenous rules (in section 4). We end the paper with some conclusion and future directions of the work.

The $\boldsymbol{r^3}$ OWL-DL ontology available at `http://rewerse.net/I5/NS/2007/r3/r3.owl` constitutes the only complete and formal definition of the $\boldsymbol{r^3}$ ontology. For the sake of readability, we have chosen to present it here using UML2 diagrams. These diagrams formally define (to the extent possible) the $\boldsymbol{r^3}$ ontology. The explanatory text that accompanies them is neither a formal definition of the $\boldsymbol{r^3}$ ontology, nor a substitute for the UML2 diagrams. As such, careful observation of the diagrams is required for full understanding of the work presented here.

[2] Examples illustrating RDF models use Turtle, omit prefix declarations, and assume the $\boldsymbol{r^3}$ namespace as the empty (':') prefix. The complete set of examples presented here may also be found in RDF/XML at `http://rewerse.net/I5/NS/2007/r3/odbase07.owl`.

2 An Ontology for Reactive Rules

An ontology for *reactive rules* restricted to different forms of *active rules* would be of limited expressivity in practice. Conditions used in ECA rules are quite often defined resorting to *logical derivation rules* (e.g. deductive rules that define intensional relations). Also, some form of *reactive derivation rules* is imperative so that symbolical events and actions with higher semantic value may be defined/derived; thus allowing reactive rules to actually express behaviour on a semantic level and not only basic low level reactions. Furthermore, integrity of a reactive system is frequently hard to express and maintain on a rule by rule basis (viz. using post-conditions): global *integrity rules* provide an additional orthogonal perspective and can be used together or independently of reactive rules allowing the detection of invalid states or reactions. Given all this, the r^3 ontology, although aiming at describing reactive behaviour, includes all these different kinds of rules, as shown in figure 1 where abstract rules (further detailed in section 4) are partitioned according to their components.

Abstract *rule components* are partitioned into consequent, antecedent and reactive, i.e. event or (trans)action, components. At this foundational level, such partition is not based on the contents of those components, but rather on a meta-level declaration (described in section 3) of what the constituent elements of a specific rule language are, and their roles in it. Such structural definition does not exclude further restriction on the contents of those components, vis-à-vis to specific rule (component) languages,

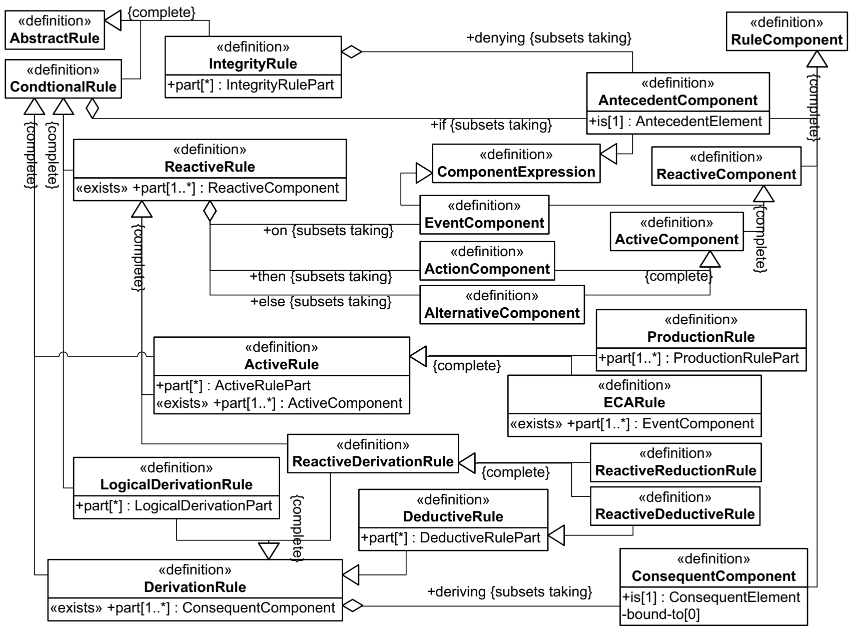

«definition» AbstractRule
{complete}
«definition» IntegrityRule
+part[*] : IntegrityRulePart
+denying {subsets taking}
«definition» RuleComponent
«definition» CondtionalRule
«definition» AntecedentComponent
+is[1] : AntecedentElement
+if {subsets taking}
«definition» ReactiveRule
«exists» +part[1..*] : ReactiveComponent
«definition» ComponentExpression
«definition» ReactiveComponent
«definition» EventComponent
«definition» ActiveComponent
+on {subsets taking}
«definition» ActionComponent
+then {subsets taking}
+else {subsets taking}
«definition» AlternativeComponent
«definition» ProductionRule
+part[1..*] : ProductionRulePart
«definition» ActiveRule
+part[*] : ActiveRulePart
«exists» +part[1..*] : ActiveComponent
«definition» ECARule
«exists» +part[1..*] : EventComponent
«definition» ReactiveDerivationRule
«definition» ReactiveReductionRule
«definition» LogicalDerivationRule
+part[*] : LogicalDerivationPart
«definition» DeductiveRule
+part[*] : DeductiveRulePart
«definition» ReactiveDeductiveRule
«definition» ConsequentComponent
+is[1] : ConsequentElement
-bound-to[0]
«definition» DerivationRule
«exists» +part[1..*] : ConsequentComponent
+deriving {subsets taking}

Fig. 1. Abstract Rules

i.e. extension of the $\boldsymbol{r^3}$ ontology is possible so that semantic coherence is maintained between the contents of the rule components and the declared nature of the associated language elements (e.g. ensuring that the content of an event component is actually an event specification).

Besides the rule components, in figure 1, also *rule parameters* are permitted, cf. figure 2, accounting for modelling semantic variations (e.g. rule priority and defeasibility) that should not have an impact on the general semantics of the different kinds of abstract rules (wrt. classification of figure 1).

Focusing on the structure of rules, and rule languages, themselves, instead of focusing on the structure of each of the rule components (or on the actual semantics induced by their contents), results in a layered approach that allows the $\boldsymbol{r^3}$ ontology to distinguish the different types of rules independently of the specific languages used in their components[3]; thus separating the semantics of rules from the semantics of their components. For instance, careful analysis of figures 1 and 2 conveys that:

– an active rule is an abstract rule that has at least one action component but no consequent component, provided all its antecedents are condition components;
– among active rules, ECA rules are distinguished from production rules according to the presence or absence of an event component;
– a derivation rule is a rule required to have at least one consequent component and optionally taking other antecedent (viz. necessity) components;
– a deductive rule is a derivation rule with symbolic consequents (i.e. views) based only on side-effect free[4] components (e.g. conditions).

Notice that the $\boldsymbol{r^3}$ ontology generalizes active rules with *alternative components* (i.e. "else"-actions). Alternative components are usually considered syntactic sugar expressible with the use of negated conditions, but given their usefulness in practice and the heterogenous nature of $\boldsymbol{r^3}$ we believe it is important to consider them. ECA rules with an alternative component (ECAA) have a clear operational semantics[5] and facilitate the modelling of workflows [15]. Further examples may be found in [11]. Regarding production rules, we are not aware of any formalization for alternative components and as such (and given that the main focus of our future work will be on ECA rules and their derivation variants), we have chosen to restrict the $\boldsymbol{r^3}$ ontology, for now, to their most usual form (viz. if-then, cf. OMG's PRR).

Among *derivation rules*, the $\boldsymbol{r^3}$ ontology distinguishes between reactive and logical derivation rules depending, respectively, on the presence or absence of a reactive component. A *reactive derivation rule*, optionally under given conditions, allows higher level symbolic events or actions (viz. occurrences, cf. figure 3) to be derived from,

[3] Naturally, this component-based approach has limitations if applied to the description of arbitrary logical rules (e.g. FOL formulas, in general, do not adhere to this component structure); nevertheless we believe it to be expressive enough to describe what is commonly understood as rules; not excluding general *formulas* as shown in figure 11.

[4] Remember that without proper extension, as mentioned before, the $\boldsymbol{r^3}$ ontology does not enforce that the content of, e.g., a condition component is actually side-effect free. It simply declares that it must be so.

[5] Given an event occurrence if the condition has no solutions, perform an alternative action.

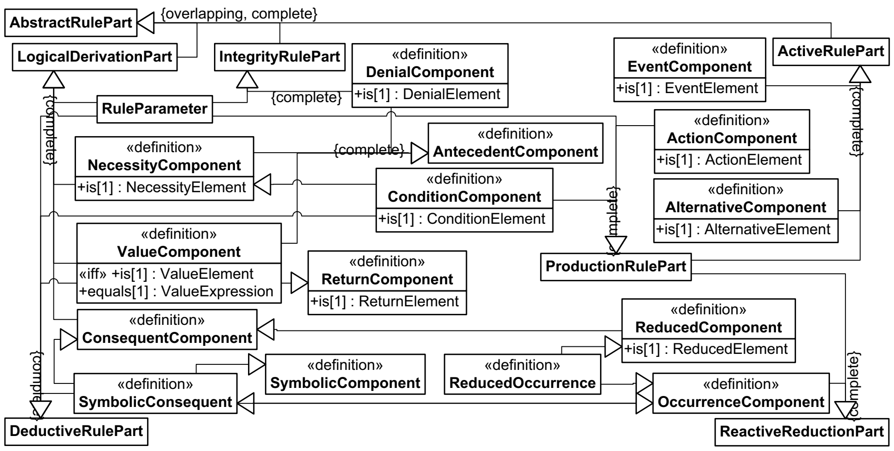

AbstractRulePart
{overlapping, complete}
LogicalDerivationPart
IntegrityRulePart
«definition» DenialComponent
+is[1] : DenialElement
«definition» EventComponent
+is[1] : EventElement
ActiveRulePart
RuleParameter
{complete}
«definition» NecessityComponent
+is[1] : NecessityElement
«definition» AntecedentComponent
«definition» ActionComponent
+is[1] : ActionElement
«definition» ConditionComponent
+is[1] : ConditionElement
«definition» AlternativeComponent
+is[1] : AlternativeElement
«definition» ValueComponent
«iff» +is[1] : ValueElement
+equals[1] : ValueExpression
«definition» ReturnComponent
+is[1] : ReturnElement
ProductionRulePart
«definition» ConsequentComponent
«definition» ReducedComponent
+is[1] : ReducedElement
«definition» SymbolicComponent
«definition» ReducedOccurrence
«definition» SymbolicConsequent
«definition» OccurrenceComponent
DeductiveRulePart
ReactiveReductionPart

Fig. 2. Rule Parts

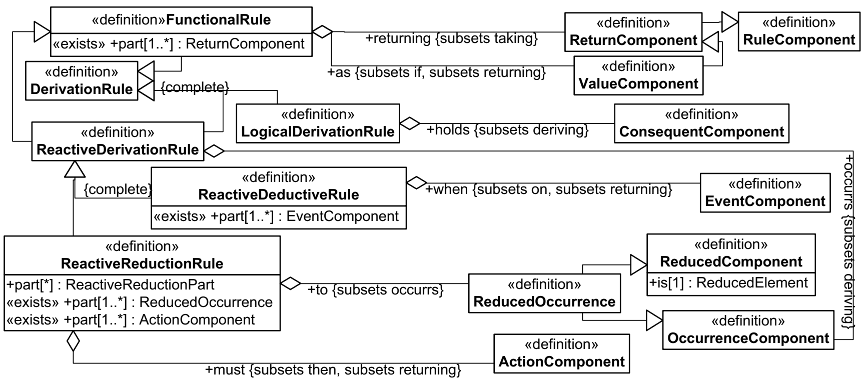

«definition»FunctionalRule
«exists» +part[1..*] : ReturnComponent
+returning {subsets taking}
«definition» ReturnComponent
«definition» RuleComponent
«definition» DerivationRule
+as {subsets if, subsets returning}
«definition» ValueComponent
{complete}
«definition» LogicalDerivationRule
+holds {subsets deriving}
«definition» ConsequentComponent
«definition» ReactiveDerivationRule
+occurs {subsets deriving}
{complete}
«definition» ReactiveDeductiveRule
«exists» +part[1..*] : EventComponent
+when {subsets on, subsets returning}
«definition» EventComponent
«definition» ReactiveReductionRule
+part[*] : ReactiveReductionPart
«exists» +part[1..*] : ReducedOccurrence
«exists» +part[1..*] : ActionComponent
+to {subsets occurrs}
«definition» ReducedOccurrence
«definition» ReducedComponent
+is[1] : ReducedElement
«definition» OccurrenceComponent
«definition» ActionComponent
+must {subsets then, subsets returning}

Fig. 3. Derivation Rules

or reduced to, other (atomic, composite or symbolic) events or actions. [2] identifies three kinds of reactive derivation rules, namely those that, under some conditions derive events from events (ECE), actions from actions (ACA) and events from actions (ACE). ECE rules have a purely deductive nature and, loosely speaking, they define views over events. ACA rules have a more operational nature and might be seen as reduction rules, rewriting higher level symbolic actions into lower level ones (similarly to instead-triggers in active databases). The intuitive idea underlying ACE rules is to declaratively express that when an action is executed (and some conditions are verified) some events occur as a derived consequence, and it may be realized through proper extension of ACA rules[6]. Usually, events derived from actions will include values that are

[6] See e.g. [3] for a formalization of a declarative reactive rule language with derivation rules, and where ACE and ACA rules are not distinguishable.

only reliably known during the action evaluation (e.g. old and new in a database update action/event). Given all this, we propose, as introduced in figure 1 and further detailed in figure 3, to partition reactive derivation rules into *reactive deductive rules* (viz. ECE) and *reactive reduction rules* (viz. ACA/ACE).

Reactive derivation rules are the subject of ongoing work and further discussion about them is not in the scope of this paper. Nevertheless, it must be stressed that reactive derivation rules are mostly uncharted territory in what concerns the (Semantic) Web. To the best of our knowledge, the only published proposal relating to this matter concerns a recent evolution of the language XChange [10], which includes reactive deductive rules[7]. As such, the proposal contained in figure 3 is introduced here mainly as a matter of completeness of the presented ontology and is to be understood as a preliminary contribution to this open research area, requiring future validation given the foundational nature of the $\boldsymbol{r^3}$ ontology.

3 Defining Reactive Rule Languages

Mainly, the $\boldsymbol{r^3}$ ontology at the current foundational (and structural) level aims at providing a Semantic Web transparent abstract syntax for reactive rule-based systems. Rule component languages are assumed to follow a term structure, using a set of functors, and functor items. Such language items are described using a meta-level of the ontology. As shown in figure 4, functors are partitioned into *language constructs* and *language symbols*, and their items are distinguished between *parameters* and construct *components*. Recursively, functor items themselves are also language symbols, i.e. functors.

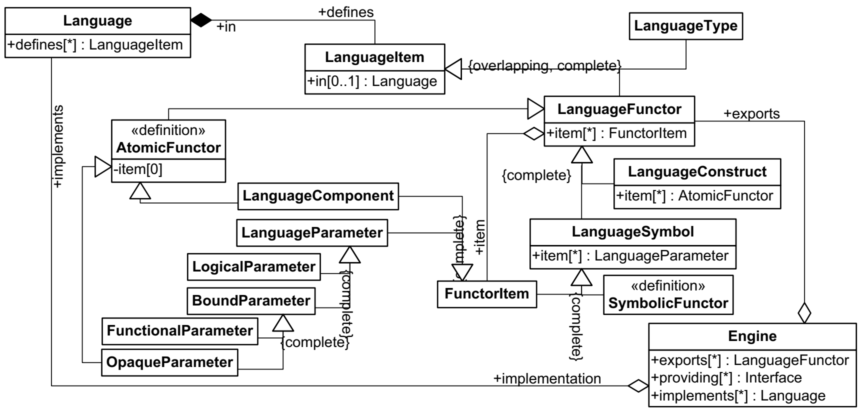


Fig. 4. Languages

[7] We strongly distinguish a reactive deductive rule that derives events (viz. occurrences); from an ECA rule performing a action (e.g. sending a message) which may induce the occurrence of events. Resorting to reactive reduction rules, an implementation of $\boldsymbol{XChange^{EQ}}$ may not need to be bound to specific (more or less ubiquitous) protocols (viz. HTTP and SOAP as suggested in [10]).

The actual operational implementation of the items of a language is to be exported by some engine. More precisely, engines evaluate constructions based on language constructs, and derive symbolic terms based on language symbols.

Language components are atomic symbols (and can only be included, as items, in a language construct). Language parameters are further distinguished between logical (i.e. input/output) and bound (i.e. input) parameters. Among input parameters, opaque parameters are distinguished from purely functional parameters. Appropriate sub-properties (viz. takes, digs, uses and binds) are introduced, in figure 5, to facilitate the declaration of functor items.

Declaratively, a functor actually represents the set of functors formed by all its ground instances (wrt. its parameters). Operationally, the semantics of a language functor (viz. construct) can not be realized unless all its input parameters are known, i.e. bound to actual values. Opaque parameters can only be used in so called opaque constructs. They account for non-atomic parameters whose values are expressed using textual or markup (sub-)languages that hide their actual structure away. A textual template where variable references are to be substituted, a snippet of code written in some scripting language that is to be interpreted, or even the literal source of a database trigger, as further detailed below, are all examples of opaque parameters.

Abstract functors, that include only abstract –functional or logical– parameters (besides components, in case of language constructs), do not require the explicit declaration of the involved variables unless for very specific cases (e.g.: quantifying variables or scoping variables implicitly quantified, and aggregators or solution modifiers).

Language constructs, cf. figure 6, are partitioned into rule, rule package and formula constructs; distinguishing native and abstract rule constructs. Only one subset of formula constructs is identified, viz. universal or existential quantifiers, but others are not excluded: e.g. conjunction, disjunction, conditionals and negation in its different variants.

Opaque formulas are allowed, and opaque native rules are restricted to purely parametric ones, i.e. no rule elements are allowed. Native rules allow the modelling of rule constructs that use textual languages that may not follow a term structure. A native rule (e.g. a database trigger) may have some functional parameters (e.g. database

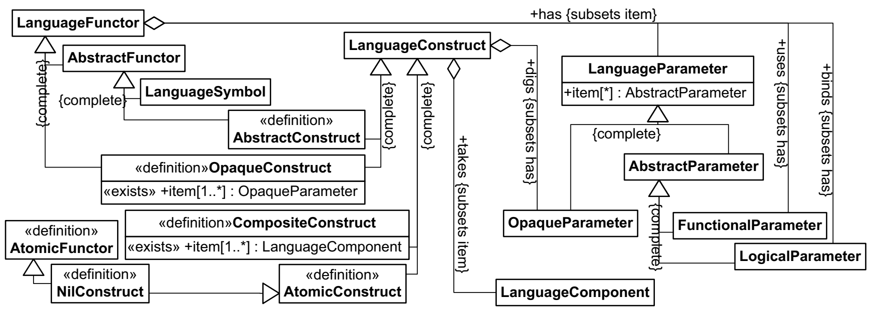


Fig. 5. Language Functors

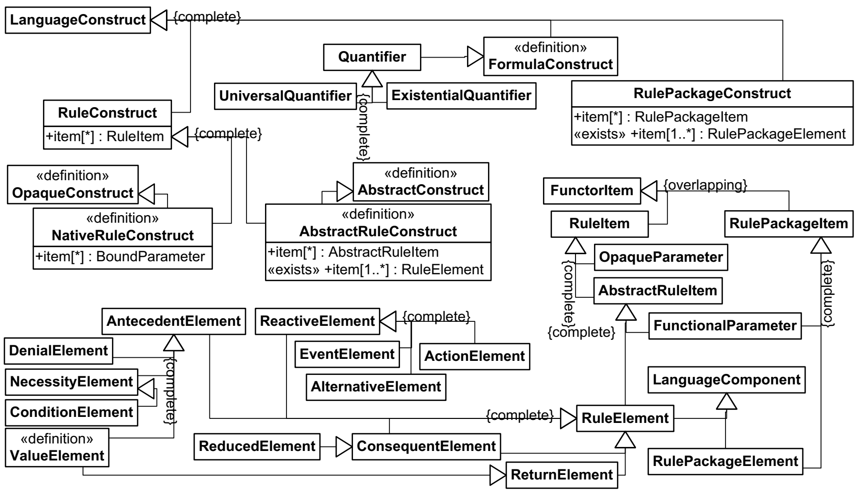


Fig. 6. Language Constructs

name, user name and password), whose atomic values are transparently used. But it is actually expressed in opaque parameters (e.g. trigger source) which hide away considerable semantic value by using “internal” (sub-)languages.

The full semantics of an opaque construct is not accessible without full knowledge of the (sub-)languages actually used inside the (consequently, non-atomic) values of such opaque parameters. In fact, the semantics of an opaque construct is known only to an engine that, cf. figure 4, exports it or implements its associated language and opaque (sub-)languages. Usually these engines will only define the semantics of such an opaque construct in operational terms, meaning that the only form of knowing it is to submit, at runtime, an actual construction to the evaluation interface provided by the engine. Nevertheless, static analysis may sometimes be possible as long as a translation interface is provided by the engine for parsing an opaque construction into an abstract one.

Example 1. For an illustrative example of an r^3 language definition[8] we resort to the current RIF Core proposal [8], more precisely to its subset dedicated to Horn rules (viz. `rif:horn`):

```
rif:ruleset a :RulePackageConstruct; :in rif:horn;
  :takes rif:rule, rif:rest.
rif:horn :defines rif:rule, rif:rest.
rif:fact a :RuleConstruct; :in rif:horn;
  :takes rif:atomic.
rif:atomic a :ConsequentElement; :in rif:horn.
rif:implies a :RuleConstruct; :in rif:horn;
  :takes rif:if, rif:then.
rif:if a :ConditionElement; :in rif:horn.
rif:then a :ConsequentElement; :in rif:horn.
```

[8] The presented definition is a partial one, namely the universal quantifier is omitted as currently it expresses only implicit quantification at the rule level, and cardinality restrictions (on the item property) should be present, in order to close the definition of the included individuals.

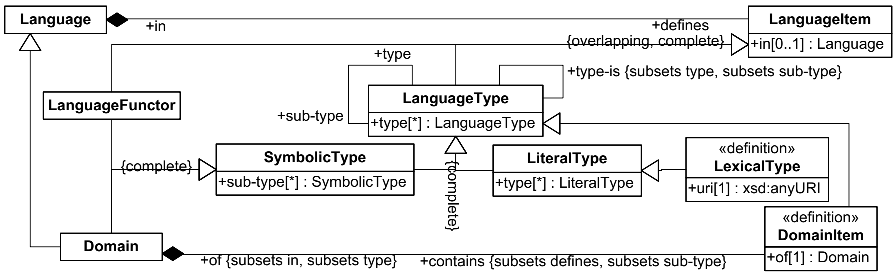


Fig. 7. Language Types

Figure [4] includes two kinds of language items, namely functors and types. Language types, as shown in figure [7], may be literal types (e.g. lexical XML Schema types identified by an URI) or symbolic types.

Every language functor implicitly defines a symbolic type; if not a functor a symbolic type is said to be a domain and is implicitly defined by a language.

A language type is considered here only as some resource that implicitly defines a set of (literal or symbolic) values. The type/sub-type relation is a containment relation: a language type contains all its sub-types and is contained in the intersection of its types.

The actual treatment of types, in the context of a general framework like [16], has not yet been considered, but it may provide, for instance, the means for a safer equality relation (e.g. xml:space preserving or not in case of lexical types). As such, the concept of language types is already included here providing the means to define (and refer to), among others, the domain of functor parameters and components, or the range of functors themselves[9].

Algebras, required in reactive rules e.g. for complex events or for process algebras in actions, may be seen as domains, as much as functors are seen as symbolic types. This leads to the definition of algebras as shown in figure [8]. Algebra constructs are called as usual operators (and their elements: arguments) and they all share the same domain.

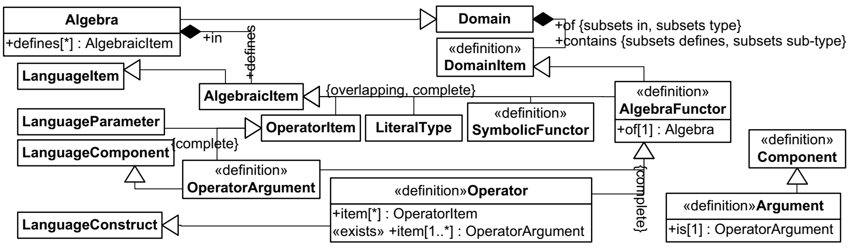


Fig. 8. Algebras

[9] Language types are also used to constrain the domain of logical variables and the range of expressions, as explained later in section [4].

Example 2. Given that any functor doubles as a symbolic type and introducing the algebraic domain of the RIF Condition language [8] (viz. `rif:condition`), we could, for instance, type-annotate the definitions of example 1 as follows:

```
rif:rule :sub-type rif:fact, rif:implies.
rif:rest :type-is rif:ruleset.
rif:if :type-is rif:condition.
rif:then :type-is rif:atomic.
rif:atomic :type rif:condition.

rif:condition a :Algebra; :in rif:core.
rif:and a :FormulaConstruct; :of rif:condition;
  :takes rif:some, rif:other.
rif:or a :FormulaConstruct; :of rif:condition;
  :takes rif:some, rif:other.
rif:exists a :ExistentialQuantifier; :of rif:condition;
  :takes rif:some.
rif:condition :contains rif:some, rif:other.
rif:equal a :FormulaConstruct; :of rif:condition;
  :binds rif:left, rif:right.
rif:condition :defines rif:left, rif:right.
```

Example 3. To further illustrate the definition of $\boldsymbol{r^3}$ languages, consider the following partial language definitions (viz. ECA-ML rule language, a minimal event algebra, some specific domain and application languages, and some "built-in" libraries).

```
 eca:rule a :RuleConstruct; :in eca:ml;
   :takes eca:event, eca:condition, eca:action.
 eca:event a :EventElement; :in eca:ml.
 eca:condition a :ConditionElement; :in eca:ml.
 eca:action a :ActionElement; :in eca:ml.
 eca:native a :RuleConstruct; :in eca:ml;
   :uses eca:lang; :digs eca:source.
 eca:opaque a :FormulaConstruct; :in eca:ml;
   :uses eca:lang; :digs eca:literal.
 eca:ml :defines eca:lang, eca:source, eca:literal.

event:sequence a :Operator; :of event:algebra;
 :takes event:first,event:next.
event:algebra :contains event:first,event:next.

travel:booking-place a :SymbolicFunctor; :of travel:domain;
 :binds travel:client,travel:flightnr,travel:seat.
travel:flight-info a :SymbolicFunctor; :of travel:domain;
 :uses travel:flight; :binds travel:date,travel:origin,travel:destination.
travel:flightnr :of travel:domain; :type travel:flight.
travel:domain :contains
 travel:client,travel:flight,travel:seat,
 travel:date,travel:origin,travel:destination.

rental:request-quotation-for-flight a :SymbolicFunctor; :in rental:application;
 :uses rental:client,rental:flight.
rental:get-client a :SymbolicFunctor; :in rental:application;
 :uses rental:client;
 :binds rental:client-name,rental:favorite-class,rental:max-price.
rental:get-available-cars a :SymbolicFunctor; :in rental:application;
 :uses rental:office,rental:date;
 :binds rental:car,rental:car-class,rental:price.
rental:client :in rental:application; :type travel:client.
rental:flight :in rental:application; :type-is travel:flight.
rental:application :defines
 rental:client-name,rental:favorite-class,rental:max-price,
 rental:car,rental:car-class,rental:price.

mail:send a :FormulaConstruct; :in mail:library;
 :uses mail:from,mail:to,mail:subject,mail:body.
mail:address :in mail:library;
```

```
  :sub-type mail:from,mail:to,rental:client,travel:client.

mail:library :defines mail:from,mail:to,mail:subject,mail:body.
text:join a :FormulaConstruct; :in text:library;
  :digs text:template; :uses text:separator.
text:replace a :FormulaConstruct; :in text:library;
  :digs text:template.
text:library :defines text:template,text:separator.
```

4 Defining Reactive Rule Constructions

The term languages modelled on the meta-level of the r^3 ontology, described in section 3, are used on a coding level to build coding resources that ultimately will define rule sets describing reactive rule-based systems. Coding resources, cf. figures 9 and 10, provide the foundations for a generic term structure (that is later used to define rules), and are partitioned between coding values (structured or not) and structure parts / constraints. Also distinguishable are coding variables, viz. references or declarations, and coding structures.

A coding structure is a language functor (or a hi-functor) gathering several structure parts (parameters or components), possibly further restricted with a set of constraints or variable declarations. A structure parameter is bound to a coding value; whereas a structure component, as further constrained in figure 10, equals a structured value (or a hi-value) whose returned value (if there is one) may still be bound to a coding value.

A hi-functor or a hi-value is a variable reference used in-place of a functor or of a component content, resp., which is to be understood in HiLog [12]. From the declarative point of view, r^3 will go no further then CL [14], i.e. its logical expressiveness is restricted to FOL. The semantics of a non-declarative, operational, “hi-construction” will be impossible to determine unless it is instantiated with a valid construction.

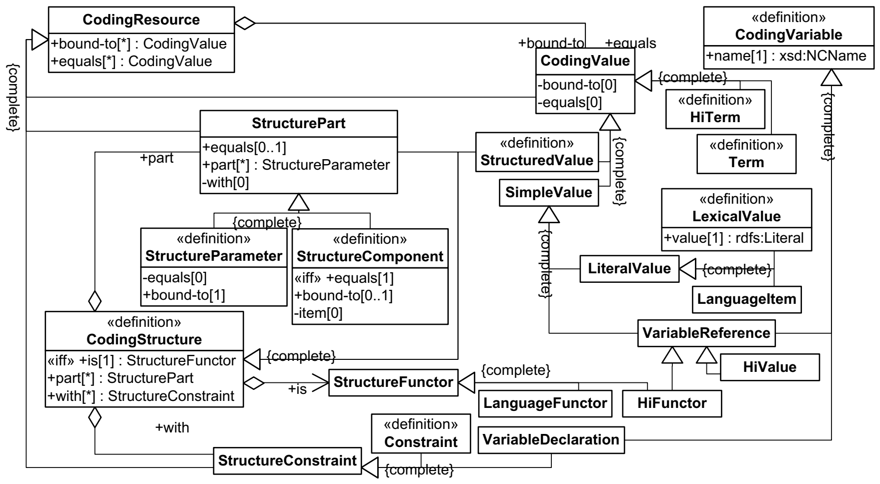


Fig. 9. Coding Resources

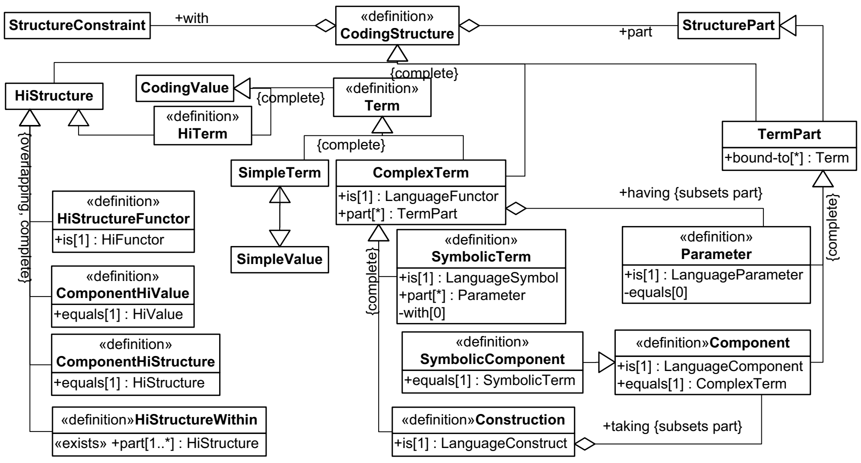


Fig. 10. Coding Terms

A coding value is either a term or a hi-term, where a term may be simple or complex. A simple term (i.e. a simple value cf. figure 10) is a literal value or a variable reference. A complex term is a coding structure, one that is a language functor and one that, as implied by the definitions in figure 10, is free of any hi-functors or hi-values. It is either a symbolic term (i.e. a language symbol without any components, constraints or variable declarations) or a construction (i.e. a language construct which, unless atomic, composes its components).

A coding structure that contains a hi-functor or a hi-value is called a hi-structure (not excluding structure parts[10]); otherwise a coding structure is either a complex term, a parameter or a component. As such, a term may be seen as a coding value –and a term part as a structure part– free of any hi-functors or hi-values. The variable referenced in a hi-functor (or hi-value) can only be bound to complex terms; possibly incomplete ones as they are to be merged with the, disjoint, set of structure parts explicitly included in the hi-structure. That said, we must stress that hi-terms are included here only as a matter of completeness of this most general level of the $\boldsymbol{r^3}$ ontology; the current main focus of the $\boldsymbol{r^3}$ ontology, and of this paper, is on terms.

Using the generic definition of a construction, and based on the language items identified in section 3, it becomes possible to identify, according to figure 11, more specific kinds of constructions: rules, rule packages, formulas and expressions (i.e. formulas without rules).

A rule is either a native rule or an abstract rule. The former has no components and is defined by its opaque source(s), whereas the latter is further partitioned based on its non-empty set of components, according to figures 1 to 3 as explained in section 2.

Any construction, given one or more input substitutions (possibly empty), when submitted to an appropriate engine, synchronously (or asynchronously in case of event

[10] Notice in figure 9 that a structure part, recursively, must be a coding structure (one without any structure constraints or components).

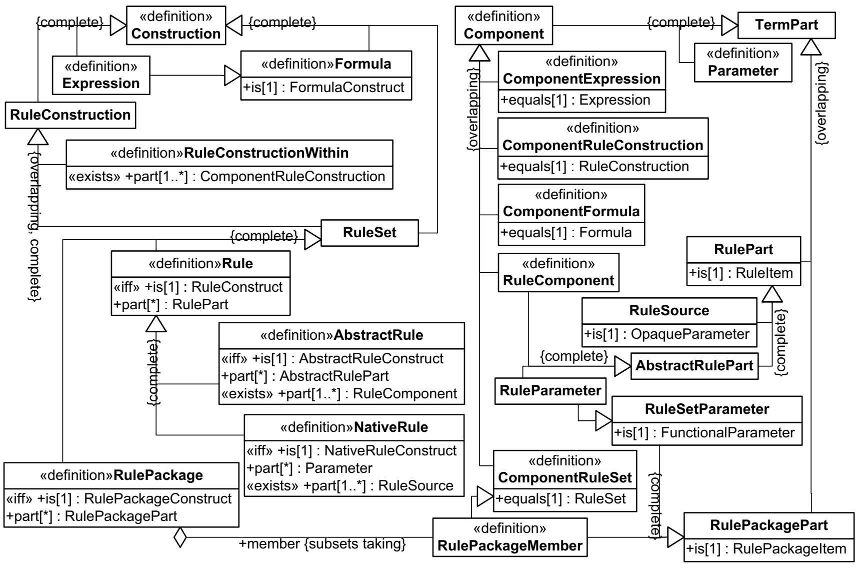

«definition» Construction
{complete}
{complete}
«definition» Expression
«definition» Formula
+is[1] : FormulaConstruct
RuleConstruction
«definition» RuleConstructionWithin
«exists» +part[1..*] : ComponentRuleConstruction
{overlapping, complete}
RuleSet
{complete}
«definition» Rule
«iff» +is[1] : RuleConstruct
+part[*] : RulePart
{complete}
«definition» AbstractRule
«iff» +is[1] : AbstractRuleConstruct
+part[*] : AbstractRulePart
«exists» +part[1..*] : RuleComponent
«definition» NativeRule
«iff» +is[1] : NativeRuleConstruct
+part[*] : Parameter
«exists» +part[1..*] : RuleSource
«definition» RulePackage
«iff» +is[1] : RulePackageConstruct
+part[*] : RulePackagePart
+member {subsets taking}
«definition» Component
{complete}
TermPart
«definition» Parameter
{overlapping}
{overlapping}
«definition» ComponentExpression
+equals[1] : Expression
«definition» ComponentRuleConstruction
+equals[1] : RuleConstruction
«definition» ComponentFormula
+equals[1] : Formula
«definition» RuleComponent
RulePart
+is[1] : RuleItem
RuleSource
+is[1] : OpaqueParameter
{complete}
{complete}
AbstractRulePart
RuleParameter
RuleSetParameter
+is[1] : FunctionalParameter
«definition» ComponentRuleSet
+equals[1] : RuleSet
{complete}
«definition» RulePackageMember
RulePackagePart
+is[1] : RulePackageItem

Fig. 11. Coding Constructions

constructions) returns several results. Each returned result contains an optional literal value and one or more output substitutions, that must be joined to one or more of the input substitutions. A construction fails if it returns no results. The returned literal values can only be used in the evaluation of the construction immediately containing the submitted construction, unless they are bound to some variable, thus extending the output substitutions. Submission of a rule set to an appropriate engine results in the rule set being loaded/activated.

A construction may further restrict its results with a set of structure constraints (viz. constraints or variable declarations, cf. figure 9). According to figure 12, some of the constraints are to be enforced during the evaluation; while others, namely post-conditions, are ensured to hold for every result of the construction. Among the former, pre-requisites are to be enforced, upon invocation, before the actual evaluation starts.

Each structure constraint defines a *constraint domain* as the intersection of its constituent domains: all possible instantiations of a particular term (it is bound to); all values returned by a value expression (it equals); or values of a particular language type. If a structure constraint has no constituent domains it trivially succeeds. Otherwise, its constraint domain must not be empty, and in case of a variable declaration it further constrains the domain of the declared variable[11]. Constraints are a generalization of the test component proposed in [16] for ECA rules (which is seen here as a constraint that equals the test expression).

[11] Additionally, a variable declaration may `rename` its variable, in order to support different naming conventions used by different opaque (sub-)languages.

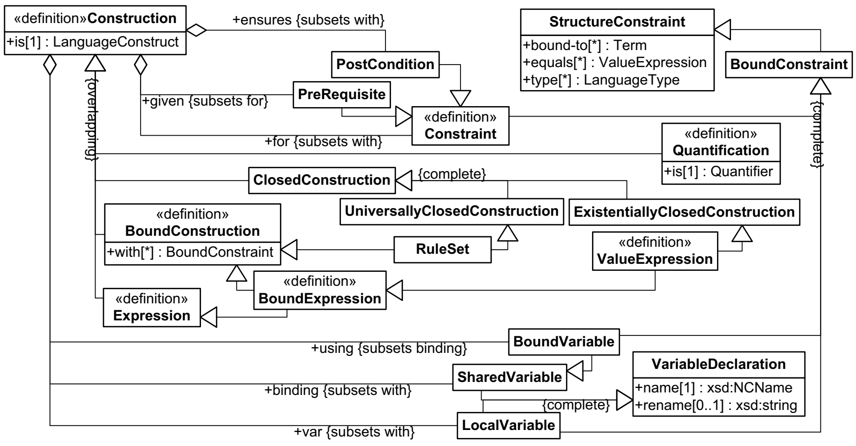


Fig. 12. Variables and Constraints

Any construction defines a scope where local variables may be declared. The communication between a sub-construction and its ancestor/sibling constructions is achieved through shared variables. Some of those shared variables may be required to be bound and a construction cannot be evaluated unless all of them are actually bound to specific values. Notice that, for efficiency, the constituents of a constraint domain, namely those based on value expressions, are expected to be incrementally evaluated thus anticipating empty constraint domain detection and variable restriction.

All variables referenced in a construction, if not explicitly declared, are implicitly declared as shared (or bound if referenced only by parameters of such nature). Furthermore, a construction implicitly inherits all the shared variables (implicit or explicit) of its descendant sub-constructions. Finally, all variables shared by bound constructions are required to be bound variables: explicit declarations are restricted accordingly in figure 12.

Although scoping of variables is allowed for any construction, their actual quantification is only achieved through explicit use of quantifiers, that quantify and scope their local variables, or through implicit quantification as induced by closed constructions, that quantify all the variables referenced within and scope them if needed (with the possible exception of bound variables they explicitly declare).

Example 4. Concluding this section, below we illustrate the usage of the r^3 ontology to define a simplified ECA rule that, as a result of some client booking a flight and requesting a car rental quotation, sends him the quotation for the cars available at the date of the flight. For this we resort to the r^3 languages included in the examples of section 3.

```
_:quotation-request-rule :is eca:rule;
 :on   [:is eca:event;     :equals _:request-for-quotation];
 :if   [:is eca:condition; :equals _:available-quotation];
 :then [:is eca:action;    :equals _:send-quotation].

_:request-quotation :is event:sequence;
```

```
  :taking [:is event:first; :equals [
    :is travel:booking-place;
    :having [:is travel:client; :bound-to [:name "Mail"]];
    :having [:is travel:flightnr; :bound-to [:name "Flight"]]]];
  :taking [:is event:next; :equals [
    :is rental:request-quotation-for-flight;
    :having [:is rental:client; :bound-to [:name "Mail"]];
    :having [:is rental:flight; :bound-to [:name "Flight"]]]].
 _:available-quotation :for _:price-ok; :is rif:and;
  :taking [:is rif:some;    :equals _:flight-info];
  :taking [:is rif:other;   :equals [:is rif:and;
  :taking [:is rif:some;    :equals _:get-client];
  :taking [:is rif:other;   :equals _:get-available-cars]]].
 _:get-client :is rental:get-client;
  :having [:is rental:client; :bound-to [:name "Mail"]];
  :having [:is rental:client-name; :bound-to [:name "Client"]];
  :having [:is rental:favorite-class; :bound-to [:name "Class"]];
  :having [:is rental:max-price; :bound-to [:name "Max-Price"]].
 _:get-available-cars :is rental:get-available-cars;
  :having [:is rental:office; :bound-to [:name "To"]];
  :having [:is rental:date; :bound-to [:name "Date"]];
  :having [:is rental:car; :bound-to [:name "Car"]];
  :having [:is rental:car-class; :bound-to [:name "Class"]];
  :having [:is rental:price; :bound-to [:name "Price"]].
 _:flight-info :is travel:flight-info;
  :having [:is travel:flight; :bound-to [:name "Flight"]];
  :having [:is travel:date; :bound-to [:name "Date"]];
  :having [:is travel:origin; :bound-to [:name "From"]];
  :having [:is travel:destination; :bound-to [:name "To"]].

 _:price-ok :equals _:check-price; :bound-to [:value "true"].
 _:check-price :is eca:opaque;
  :using [:name "Price"], [:name "Max-Price"; :rename "MaxPrice"];
  :having [:is eca:lang; :bound-to [:value "http://www.w3.org/XPath"]];
  :having [:is eca:literal; :bound-to [:value "$Price <= $MaxPrice"]].
 _:send-quotation :is mail:send;
  :var [:name "Text"; :equals _:quotation-message];
  :having [:is mail:from; :bound-to [:name "Rental-Mail"]];
  :having [:is mail:to; :bound-to [:name "Mail"]];
  :having [:is mail:subject; :bound-to [:value "Car Rental Quotation"]];
  :having [:is mail:body; :bound-to [:name "Text"]].
 _:quotation-message :is text:replace;
  :var [:name "Priced-Cars";
    :equals [:is text:join;
      :using [:name "Car"], [:name "Price"];
      :having [:is text:template; :bound-to [:value "|Car|=|Price|"]];
      :having [:is text:separator; :bound-to [:value ", "]];]];
  :using [:name "Client"], [:name "Flight"], [:name "Date"], [:name "To"];
  :having [:is text:template; :bound-to [:name "QuotationTemplate"]].
```

5 Conclusion and Future Work

The r^3 ontology provides a foundation to describe both reactive rules and reactive rule languages. The latter is accomplished on a meta-level where not only rule languages themselves but also other supporting languages used in rule components can be described. The former is provided following a term-based compositional approach that allows not only the use of different languages for different rule components (e.g. event, condition, action), but also the composition of different languages in a single component possibly using algebraic languages, thus accounting for language heterogeneity.

The r^3 ontology recognizes that full expressivity of reactive behavior can not be achieved without the help of derivation rules. As such it includes a proposal for reactive derivation rules (complementing the more traditional active rules, viz. ECA and production rules) and also logical derivation rules. Additionally, it recognizes the importance of reliability for reactive systems and contemplates global integrity rules.

Furthermore, the $\boldsymbol{r^3}$ ontology contributes for the clarification of concepts through formal definition of an RDF vocabulary for describing rule-based reactive behaviour; notably establishing a clear distinction between reactive, active and deductive rules, consistently with the terminology traditionally used in the Active Databases field. As obvious as it may seem, it should be stressed that the formal definition of such a (Semantic Web transparent) vocabulary is not to be confused with the formalization of a semantics for reactive rule languages; the former is the subject of this paper and provides the basis (as an abstract syntax) for the definition of the latter (which is out of scope here).

Last, it is worth mentioning that the work here presented builds upon previous concrete proposals of the $\boldsymbol{r^3}$ ontology that provided the basis for the current implementation of the $\boldsymbol{r^3}$ prototype [1] available online [23]. In this prototype a previous version of the $\boldsymbol{r^3}$ ontology has already been used to model several component languages. Namely, the languages Xcerpt for queries, XChange for events and actions of updates of XML data, Prova, XQuery/XPath and HTTP. This work actually lead to the inclusion of these `Languages` (as fully functional `Engines`) in the current version of the $\boldsymbol{r^3}$ prototype. Details about the previous definition and `implementation` of these `Languages` may be found in [1]. This experimental work on defining `Languages` has brought relevant contributions for the $\boldsymbol{r^3}$ ontology and must continue to be pursued and extended to other languages like, e.g.: XSLT, the algebraic language for actions described in [6], the $\boldsymbol{XChange^{EQ}}$ [10] or ruleCore [20] event languages, or even the SPARQL algebra.

At the current foundational level, the $\boldsymbol{r^3}$ ontology mainly defines a Semantic Web transparent abstract syntax for reactive rules. As usual for most abstract syntax, it allows invalid constructions without a defined semantics (e.g. infinite terms). This abstract syntax must be validated against and complemented with a formal semantics definition. Achieving such a formal definition, together with the re-implementation of the $\boldsymbol{r^3}$ prototype according to it, constitutes our major goal for the immediate future.

The future work on the $\boldsymbol{r^3}$ semantics and prototype is to be focused mainly on ECA and reactive derivation rules, although not discarding consistent integration with the other types of rules (particularly the evolution of RIF [8] is to be followed as closely as possible). A very important matter needs to be carefully considered: a solution grouping mechanism is mandatory for actions (as careful analysis of example 4 shows). Whether such grouping is achieved through the use of grouped aggregations, or resorting to solution modifiers like project and distinct, or left out for action languages is still an open matter under discussion at the time of this writing. As such we have chosen not to include any proposal on this issue here. On the other hand, types and hi-terms are a matter to be taken conservatively and may be postponed until a stable version of the prototype is available.

Considering the extension of the $\boldsymbol{r^3}$ ontology to higher abstraction levels (e.g. rule languages, domain/application languages, event languages, event algebras, process algebras) is also to be pursued in close cooperation with the MARS project [22]. Furthermore, the $\boldsymbol{r^3}$ ontology implicitly extends and generalizes the previously proposed general ECA framework [2,16,17] (e.g. derivation rules, solution constraints and rule constructions) and calls for a revision of the ECA-ML markup [16] and of the general framework. Any revision of the ECA-ML markup should: consider an homogenous markup for logical variables, and be based upon the formal specification of the (revised) framework. Given the r^3 ontology (as an adequate abstract syntax for the framework), such a (fully) formal specification should now be possible to achieve without resorting to general markup guidelines and principles.

Although not directly related, given its ontology based approach, to current textual or markup based proposals for concrete Web ECA languages (e.g. [9,19,10]), r^3 undoubtedly aims at modelling most (if not all) of them. As such, given a formal semantics of r^3, actual demonstration of its adequacy to this purpose is also to be pursued.

References

1. Alferes, J.J., Amador, R., Behrends, E., Bry, F., Eckert, M., Franco, T., Fritzen, O., Grallert, H., Knabke, T., Krippahl, L., May, W., Pătrânjan, P L., Schenk, F., Schubert, D.: Completion of the prototype scenario. I5-D7, CENTRIA, Universidade Nova de Lisboa (2007), http://rewerse.net/deliverables.html
2. Alferes, J.J., Amador, R., Behrends, E., Fritzen, O., Knabke, T., May, W., Schenk, F., Schubert, D.: Reactive rule ontology: RDF/OWL level. I5-D6, CENTRIA, Universidade Nova de Lisboa (2007), http://rewerse.net/deliverables.html
3. Alferes, J.J., Banti, F., Brogi, A.: An Event-Condition-Action Logic Programming Language. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, Springer, Heidelberg (2006)
4. Alferes, J.J., Brogi, A., Leite, J.A., Pereira, L.M.: An evolvable rule-based e-mail agent. In: Pires, F.M., Abreu, S.P. (eds.) EPIA 2003. LNCS (LNAI), vol. 2902, Springer, Heidelberg (2003)
5. Angele, J., Boley, H., de Bruijn, J., Fensel, D., Hitzler, P., Kifer, M., Krummenacher, R., Lausen, H., Polleres, A., Studer, R.: Web Rule Language (WRL). In: W3C (September 2005) (submission), http://www.w3.org/Submission/2005/SUBM-WRL-20050909/
6. Behrends, E., Fritzen, O., May, W., Schenk, F.: Combining ECA Rules with Process Algebras for the Semantic Web. In: RuleML 2006, IEEE, Los Alamitos (2006)
7. Boley, H., Grosof, B., Sintek, M., Tabet, S., Wagner, G.: RuleML Design. RuleML Initiative (August 2006), http://www.ruleml.org/
8. Boley, H., Kifer, M.: RIF Core Design. Working Draft, W3C (August 2007), http://www.w3.org/TR/2007/WD-rif-core-20070330
9. Bonifati, A., Braga, D., Campi, A., Ceri, S.: Active XQuery. In: ICDE 2002, IEEE, Los Alamitos (2002)
10. Bry, F., Eckert, M.: Rule-Based Composite Event Queries: The Language XChangeEQ and its Semantics. In: Marchiori, M., Pan, J.Z., de Sainte Marie, C. (eds.) RR 2007. LNCS, vol. 4524, Springer, Heidelberg (2007)
11. Bry, F., Eckert, M., Pătrânjan, P.-L., Romanenko, I.: Realizing Business Processes with ECA Rules: Benefits, Challenges, Limits. In: Alferes, J.J., Bailey, J., May, W., Schwertel, U. (eds.) PPSWR 2006. LNCS, vol. 4187, Springer, Heidelberg (2006)
12. Chen, W., Kifer, M., Warren, D.S.: HILOG: A Foundation for Higher-Order Logic Programming. Journal of Logic Programming 15(3), 187–230 (1993)

13. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C (2004)(submission), http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/
14. International Organization for Standardization. Common Logic (CL): a framework for a family of logic-based languages, volume ISO/IEC FDIS 24707. ISO (May 2007), http://common-logic.org/
15. Knolmayer, G., Endl, R., Pfahre, M.: Modeling processes and workflows by business rules. In: Business Process Management, Models, Techniques, and Empirical Studies, pp. 16–29. Springer, Heidelberg (2000)
16. May, W., Alferes, J.J., Amador, R.: Active rules in the Semantic Web: Dealing with language heterogeneity. In: Adi, A., Stoutenburg, S., Tabet, S. (eds.) RuleML 2005. LNCS, vol. 3791, Springer, Heidelberg (2005)
17. May, W., Alferes, J.J., Amador, R.: An ontology- and resources-based approach to evolution and reactivity in the Semantic Web. In: Meersman, R., Tari, Z. (eds.) ODBASE 2005. LNCS, vol. 3761, Springer, Heidelberg (2005)
18. Object Management Group. Semantics of Business Vocabulary and Business Rules (SBVR). OMG August (2006), http://www.omg.org/cgi-bin/doc?dtc/2006-08-05
19. Papamarkos, G., Poulovassilis, A., Wood, P.T.: Event-condition-action rules on RDF metadata in P2P environments. Computer Networks: The International Journal of Computer and Telecommunications Networking 50(10), 1513–1532 (2006)
20. Seiriö, M., Berndtsson, M.: Design and implementation of an eca rule markup language. In: Adi, A., Stoutenburg, S., Tabet, S. (eds.) RuleML 2005. LNCS, vol. 3791, Springer, Heidelberg (2005)
21. Spreeuwenberg, S., Gerrits, R.: Business Rules in the Semantic Web, Are There Any or Are They Different? In: Barahona, P., Bry, F., Franconi, E., Henze, N., Sattler, U. (eds.) Reasoning Web. LNCS, vol. 4126, Springer, Heidelberg (2006)
22. MARS: Modular Active Rules for the Semantic Web. DBIS, Institute for Informatics, Georg-August-Universität Göttingen. http://rewerse.net/I5/MARS/
23. Resourceful Reactive Rules(r^3). CENTRIA, Universidade Nova de Lisboa, http://rewerse.net/I5/r3/

Heuristics for Constructing Bayesian Network Based Geospatial Ontologies

Sumit Sen[1,2] and Antonio Krüger[2]

[1] Deptt of Computer Science,
IIT Bombay, Mumbai 400 076, India
[2] Institute for Geoinformatics
University of Münster,
Robert Koch Str. 26, 48149 Münster, Germany
sumitsen@uni-muenster.de, kruegera@uni-muenster.de

Abstract. Bayesian Network based ontologies enable specification of partial relations between concepts as an advantage over conventional ontologies, based on description logic. In the context of geospatial ontologies such specifications facilitate encoding relations between action and entitiy concepts. This paper presents a case study of transportation ontologies based on traffic code texts of two different countries. We construct ontologies of both geospatial entities and actions using the BayesOWL approach. Thereafter we employ heuristics based on verb-noun co-occurence evidences, available from analysis of formal texts, to construct linkages between the two types of concepts. This approach enables high recall and precission for querries on concepts and enables rich inferences such as most similar and disimilar concepts. The results of our experiments are verified with human subjects testing. Such heuristics-based-probablisitic approaches to geospatial ontology specification and reasoning can be utilized for concept mapping within and across geospatial ontologies as well as to quanitfy the naming hetrogeinities in two given ontologies.

1 Introduction

Ontologies are used to specify and obtain agreements about conceptualizations about entities represented in information systems [1]. Ontologies are used as tools to improve semantic interoperability between information systems [2]. Geospatial ontologies or geo-ontologies have been employed to obtain similar results in the geospatial domain as well [3], [4]. However several issues remain unresolved in the context of ontologies in general and geospatial ontologies in specific. Handling of partial knowledge about entity classes and their relations is one such aspect.

Conventional ontology languages are based on crisp logic such as Description Logic (DL) and thus cannot handle incomplete or partial knowledge about an application domain. Uncertainties are ubiquitous in all domains and all aspects of ontologies. For example, in a given ontology about transportation networks, besides knowing that "*Highway* is a subclass of *Road*" and "*Street* is a subclass of *Road*", we would like to express the probability that an instance of *Road* is also a *Street* (or in the other case, is an instance of *Highway*). Such information is important and sometimes critical for practical reasoning tasks in ontologies as shown by Ding et al [13]. In the context of geospatial ontologies the flexibility provided by the probablisitic framework of ontologies, such as BayesOWL [13] enables encoding partial knowledge about linkages between concepts relating to geospatial entities and actions. Such knowledge is in the form of affordance values that may be determined by various techniques including analysis of formal texts [6]. This paper outlines a heuristics based approach of specifying the linkages between geospatial entities and actions using the BayesOWL approach [13]. We discuss the results of the approach along with its applications in the area of matching concepts within and across ontologies. We also demonstrate that our approach allows us to quanitify interoperability between two given ontologies.

1.1 Geospatial Ontologies as Knowledge Representation Tools

Existing literature in geographical information science points out the significance of geospatial ontologies as tools to represent conceptualizations in the geospatial domain. Such knowledge representation tools are mostly used to resolve semantic differences and promote interoperability between applications across information communities [7].

Agarwal [5] has discussed that a unified approach to ontology specification in the geospatial domain does not exist. Different approaches including the approaches of formal ontologies [8] and algebraic approaches [9] [10] have evolved in parallel to the conventional approaches of Description Logic (DL) based specifications. Geospatial ontology engineering has been also proposed to enable a supportive environment for knowledge representation in the geospatial domain[11] . However the challenges for geospatial ontologies as tools of knowledge representation remain unresolved to a large extent. The primary questions that need to be answered include.

(i) Gomez Perez [12] has stated that the number of ontologies specified is not large enough for their use in practical and industrial scale applications. This is true for the geospatial domain and practically verified ontologies are still to be produced. In their absence it is impossible to verify their utility and hence their contributions to semantic interoperability
(ii) In a similar point of view it has been discussed that the tools and principles of ontologies are still viewed with skepticism even after years of research. Agarwal [5] has pointed that geographic concepts and categories have inherent indeterminacy and vagueness especially that emerge from human reasoning and conceptualization. It is therefore unlikely that the semantic ambiguities can be resolved without accounting for the uncertainty factor.
(iii) Geospatial ontologies have either looked at geographic space either from the point of view of the geospatial entities with it or from that of geospatial actions. A unified view which incorporates knowledge of geospatial actions in ontologies of geospatial entities and which treats both these components of knowledge as first class citizens, is necessary. Kuhn[6] advocates the inclusion of actions and affordances in geospatial ontologies.

Geospatial ontologies are in need of innovative approaches to ensure their practical use. In order that geospatial conceptualizations can be encoded in ontologies, emerging techniques in ontological specifications and knowledge representation need to be adapted and experimented in the geospatial domain. These include probabilistic ontologies [13] and inclusion of knowledge about geospatial actions and their hierarchies [14]. The use of probabilistic geospatial ontologies also enables stochastic reasoning, which is capable of producing quantitative results to evaluate the similarity of concepts across ontologies as well.

1.2 Ontologies of Geospatial Entities and Actions

While ontologies of geospatial entities is similar to conventional ontologies of substances or endurants[15] On the other hand, the term Action-driven ontologies was first coined by Gilberto Câmara *et al* [16] to address the intentionality perspective in categorizing the *fiat* and *bona-fide* objects in geospatial ontologies. The view on geospatial ontologies that is advocated by them, states

"geographical space" as "a system of entities and a system of actions"

This view of geographical space has been shared widely across the geospatial domain by various groups. The notion of functions of geospatial entities should be considered analogous to that of actions, and should be viewed in the broader sense as it appears in ontology literature. Timpf [17] maintains that geography activity models help to disambiguate semantics of data in a particular domain. Kuhn [6] postulates that geospatial ontologies need to be designed with a focus on human actions and activities in geographical spaces. He advocates simple steps of identifying human actions and activities in the domain space and modelling them with respect to the domain entities. The assumption underlying this approach is that increasingly complex activities are a precursor to increasingly complex conceptualizations of the environment and that such a paradigm ensures that ontologies are both conciseness and relatedness to human activities. Thus as thumb rule concepts of functions are less in number than concepts of entities.

Mainstream efforts in geospatial ontologies, and ontologies in general have concentrated on so called substantial entities [15] and hence static taxonomies rather than functions or roles. Worboys [18], states

"*The large majority of current systems for handling geospatial information are static, concentrating on a single temporal snapshot, usually the current state.*"

Johansson [19] has discussed the general role of functions in classification of entities. Ontology engineering by making use of functionality of engineering artifacts in the manufacturing domain has been discussed by Kitamura *et al* [20]. It shown that an ontological framework for functional knowledge helps interoperability of design data in the form CAD/CAM diagrams. For our study we use analysis of traffic code texts texts similar to the work of Kuhn [6] and hence restrict the scope of functions or affordances of road network entities to those defined with in the formal text. We discuss the analysis of texts in section 3.2.

2 Probabilistic Ontologies and Stochastic Inferences

Conventional DL based ontologies such as OWL ontologies are characterized by the following.

1. *Expressivity:* DL based constructs are highly expressive, enabling rich and complex descriptions of domain concepts.
2. *Automated Reasoning*: Based on logic based reasoning, concepts can be checked for consistency and to evaluate Subsumption.
3. *Compositionality*: The previous two characteristics enable creation of new concepts based on existing ones by combining concepts and properties.

Such characteristics are also accompanied by certain deficiencies such as

1. Inability to express probabilistic relationships similar to the links between functions and entities discussed above. These could be accommodated in the various extensions suggested such as P-CLASSIC [21], Fuzzy DL [22] besides others [23].
2. In a practical case it is often that one concept in an ontology does not exactly match another concept in a second ontology. The match is usually a combination of multiple concepts, which is discussed as the one-many problem [24] and requires the capability to match the most similar concept.

We discuss probabilistic ontologies, which are expressed as Bayesian Networks (BN). The advantage of such ontologies is that a new logic framework is not required. BN based ontologies also allow stochastic inferences such as most similar and most dissimilar concepts.

2.1 Ontologies as Bayesian Networks

PR-OWL [25] and BayesOWL [13] are two approaches that use BN based representation of ontologies. Of these BayesOWL provides an approach for specification and reasoning.

Ding *et al.* [13] developed a mechanism of expressing OWL ontologies as Bayesian networks termed as BayesOWL. The important steps to construct such ontologies are as below:

Construction of the Directed Acyclic Graph (DAG): The entity classes to be used are listed first and the topmost (most universal) concept is added to the top of the DAG as a node. Child concepts of this concept are added below the parent concept as individual nodes and the complete DAG is created by constructing the links. Each node has only 2 states (True, False)

Regular Nodes and L Nodes: The nodes created above are called Regular nodes. There are another category of nodes called L Nodes which help in constructing Union, Intersection, Disjoint and Equivalent relationships. Since we do not use any of these relationships in our ontologies we shall ignore construction of L Nodes.

Allocating conditional probabilities: Regular nodes (other than the top node) have one conditional probability value each for its parent node. It is suggested that such conditional probability values are learnt from text classification techniques. We use the relatedness values from WordNet similarity modules [26] to derive these values.

Applying IPFP iterations to impose P Space: Finally with given (Conditional Proabability Table) CPT values it is important for the network to learn the real values given the probability constraints to arrive at a condition where all LNodes are true. This is achieved by a iterative proportional fitting procedure [27]. In case there are no L Nodes to be considered this iterative step can be overlooked.

2.2 Reasoning Tasks in Bayesian Network Ontologies

In the most general form, a BN of n variables consists of a DAG of n nodes and a number of arcs. Nodes Xi in a DAG correspond to variables, and directed arcs between two nodes represent direct causal or influential relation from one node to the other. The uncertainty of the causal relationship is represented locally by the conditional probability table (CPT) $P(X_i|\pi_i)$ associated with each node X_i, whereπ_i is the parent node set of Xi. Under a conditional independence assumption, the graphic structure of BN allows an unambiguous representation of interdependency between variables, which leads to one of the most important feature of BN: the joint probability distribution of $X = (X_1,\ldots,X_n)$ can be factored out as a product of the CPTs in the network (named \the chain rule of BN"):

$$P(X = x) = \prod_{i=1}^{n} P(X_i \mid \pi_i) \tag{1}$$

With the joint probability distribution, BN supports, at least in theory, any inference in the joint space although probabilistic inference in a general DAG is NP-hard [Cooper, 1990]. It is interesting to note the similarity between our ontologies and the requirement of Directed Acyclic Graphs to construct BNs.

The principle reasoning tasks in our Bayesian network are based on computation of joint probability distributions and utilize the three methods suggested by Ding *et al.* [13]. These are:

- *Concept Satisfiablity*: if a concept based on certain states of given nodes in the network can exist. This is defined by verifying if P (e|t) = 0, where e is the given concept and *t* indicates the truth condition of other concepts.
- *Concept Overlap*: the degree of overlap between a given concept *e* and any other concept in the network is determined by P (e|C,t).
- *Concept similarity:* The advocated measure of similarity is based on Jaccard coefficient provided by Rijsbergen [28] as

In our case this translates to

$$\begin{aligned} MSC(e,C) &= P(e \cup C) / P(e \cap C) \\ &= P(e,C) / (P(e) + P(C) - P(e,C)) \\ &= P(C|e) * P(e) / (P(e)+ P(C)-P(C|e)* P(e)) \end{aligned}$$

For more detailed explanations of these reasoning tasks refer Ding et al. [13]

3 Case Study: Ontologies from Traffic Code Texts

Traffic code texts such as the Highway Code of UK[1] (HWC) and the New York Driver's Manual[2] (NYDM) are examples of formal texts, which not only mention the entities in a road network but also specify the permissible actions in the respective geographic jurisdiction. Kuhn [29] has advocated the extraction of ontologies from such formal texts. Our case study involves the extraction of such ontologies from each

[1] www.highwaycode.gov.uk/
[2] http://www.nydmv.state.ny.us/dmanual/

of these traffic codes. We extract most frequently occurring entities and construct hierarchies of such entities. We also extract most frequently occurring actions in relation to these entities and construct hierarchies of actions as well. A further text analysis provides with co-occurrence values of entity-action pairs, which are used to establish linkages between entities and their actions.

In this section we discuss the extraction of probabilistic ontologies based on the text analysis. We also discuss the inferences obtained from such ontologies as opposed to conventional ontologies.

3.1 Extraction of Entity Concepts and Action Concepts

The steps listed in § 2.1 are used to construct the BN based ontologies. The important constituents required for these are extracted from the text as follows.

1. Both texts are subjected to a Part Of Speech (POS) analysis which not only analysis the part of speech but also provides the sense of the words [30].
2. The most frequently occurring entities are used to construct a hierarchy of geospatial entities using hypernyms relations of noun terms from WordNet lexicon [30].
3. Similarly a hierarchy of geospatial action terms are used to construct the hierarchy of actions. Hypernym relations between verbs are used to construct such hierarchies.
4. WordNet-similarity modules [26] are used to extract the conditional probabilities between class and subclass relations in the two hierarchies. The CPTs thus obtained allow us to construct individual BayesOWL ontologies of entities and actions separately as shown below in figure 1 and 2.

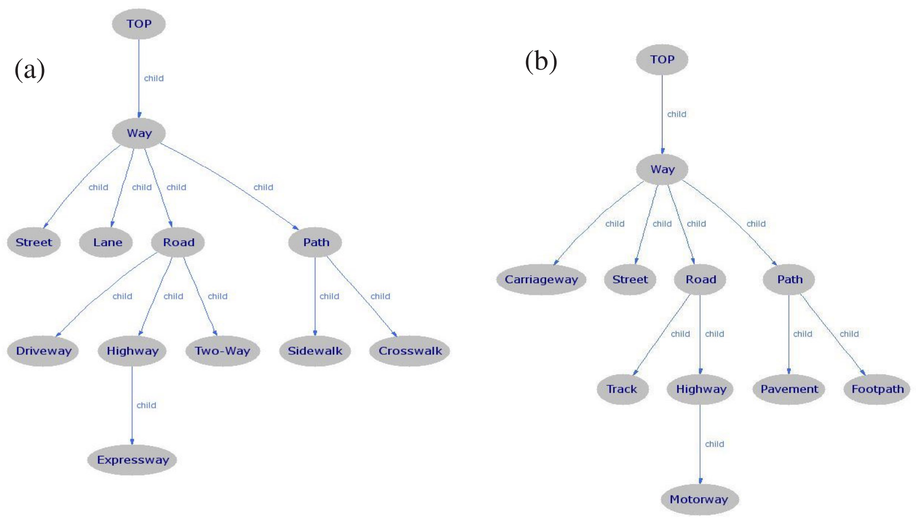


Fig. 1. DAG of entity concepts extracted from the (a) NY Driver Manual text and (b) the UK Highway code text

It is important to note that many concepts in one entity concept graphs do not reoccur in the other graph. However in the context of action concepts all the concepts occurred in both texts. The natural motivation for linking the two graphs, which emerges from these graphs are as follows

1. to match the most similar and most disimilar concepts for the concepts from the second ontology where these concepts are missing or named differently.
2. to evaluate the performance of the machine based mappings in cases where the names of the concepts are same across ontologies.

We make an important assumption based on Kuhn [29] that geospatial actions form basic concepts of the geospatial domain, to which entity concepts could be linked.

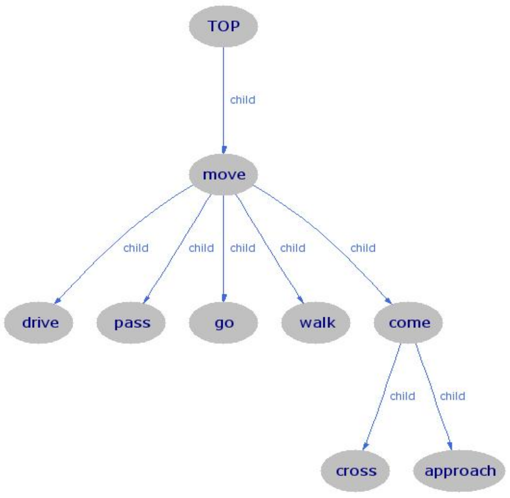


Fig. 2. DAG of action concepts extracted from the both traffic code texts

The CPTs of these two ontologies cannot be represented here for lack of space but it is important to state that these form an important part of the ontology itself. Also it is important to note that many of the nodes show “0+” value for the state ‘True’. These values are small but not equal to zero (A zero value would indicate concept unsatisfiablity but a very small value would indicate that such concepts can still be satisfied but very rarely).

3.2 Heuristics: Linking Entities and Actions

As discussed in section 2.3 we know that noun-verb co-occurrences are important sources for extraction of comprehension of the affordance of a road network entity. Thus a sentence from the NYDM which says:

You may never make a U-turn on a limited access expressway, even if paths connect your side of the expressway with the other side.

means, *limited access expressways* do not afford *U-turns*, although the physical affordance still exists. For our paper we shall not distinguish between physical and social/legal affordances but recognize that such distinctions can be made.

The co-occurrence analysis of the two texts gives us co-occurrence/occurrence index values as shown in table 1 below. These values are indicative of the affordance each of the entities listed have for the corresponding actions. The zero values in the table indicate non-affordances where as blanks indicate that no information was available to ascertain affordance or non-affordance.

Kuhn [6] reports similar results about the German traffic code, although his results are rather deterministic (do not use a quantitative value to represent the affordance) and do not report non-affordances. We extend the work of Kuhn [6] by adding such information and propose to include such knowledge in the geospatial ontologies.

The level of automation in terms of machine-based analysis of the affordance is greater in our case but is also accompanied with careful manual inspection. We confirm the observations of Kuhn [6] that manual extraction is tedious but at times most effective.

Kuhn [6] has also reported hierarchies of actions in the German traffic code and this includes the notion of entailment of actions. Four notions of entailments commonly seen in verbs have been reported in linguistics [31]. These include:

- Troponymy, which express super-sub concept relationships among verbs and hence the actions.
- Inclusion, which expresses *part-of* relationships between two actions, implying that one action is a part of the other.
- Pre-requisiteness, where one action acts as a pre-requisite for the second but not necessarily causing the second action.
- Causality, when one action initiates (or is the cause) of the second action

These relations are very useful to probe nested and sequential affordances. For example the complex action *to overtake* is composed of the actions *drive, approach* and *pass.* The affordance of another vehicle to be *passed* is only realized when the affordance of *approachability* has been used. Much more complex actions can be explored with the notion of such sequential affordances.

At the same time, by using *is-a* relations we can express hierarchies of action concepts. As done in Figure 2 discussed earlier. The affordance of the lower concepts such as *walk, drive* etc. entails the affordance of *movement* and is nested inside the former.

Table 1. Afordances of road network entities based on co-occurrence analysis of the two traffic code texts

HWC	Street	Road	Footpath	Motorway	Lane	Way	Path	Crosswalk	Expresswa
move	0.015	0.049	-	0.012	0.107	0.035	-	-	-
walk	-	0.026	0.056	0.000	-	-	-	-	-
drive	0.057	0.062	0.000	0.069	0.000	-	-	-	-
enter	-	0.025	-	-	0.000	0.020	-	-	-
stop	0.010	0.075	-	0.000	0.000	0.051	-	-	-
be	0.014	0.215	0.006	0.028	0.061	0.033	0.014	-	-
cross	0.029	0.135	-	0.000	0.024	0.067	0.020	-	-

Table 1. (*continued*)

turn	0.038	0.059	-	-	0.042	0.041	-	-	-
wait	-	0.040	-	0.000	0.009	0.031	-	-	-
approach	0.022	0.052	-	0.016	0.065	0.045	0.023	-	-
go	-	0.021	-	-	0.063	-	-	-	-
pass	-	0.038	-	-	0.032	0.012	0.017	-	-
NYDM									
move	0.026	0.032	-	-	0.107	-	0.032	-	-
walk	-	0.010	-	-	-	-	-	-	-
drive	0.020	0.061	-	-	0.056	-	-	-	0.047
enter	0.025	0.048	-	-	0.077	0.041	-	0.053	0.064
stop	0.019	0.048	-	-	0.038	0.026	-	0.059	0.026
be	0.011	0.068	-	-	0.089	0.026	0.004	0.009	0.024
cross	0.061	0.033	-	-	0.017	0.071	-	0.030	
turn	0.037	0.080	-	-	0.094	0.051	0.029	0.018	0.008
wait	0.040	-	-	-	0.009	0.059	-	-	0.029
approach	0.015	0.060	-	-	0.034	-	-	-	0.026
go	0.020	0.029	-	-	0.030	0.051	-	-	0.017
pass	0.044	0.039	-	-	0.130	0.025	-	0.014	0.013

Given these values, links are established between concepts of entities and actions. Affordance values are used for Conditional Probababilities (CPs) for these links.

3.3 Inferences from Linked Ontologies

Inferences within BayesOWL as described in section 2.2 can be applied to each probabilistic geo-ontologies of the HWC & the NYDM. The DAGs for the two traffic-code ontologies are shown in Figure 3 and 4 below. Stoichastic reasoning upon these Bayesian Networks allow us to obtain the most similar concepts with in the same ontology and the degree of overlaps. This is particularly useful if one needs to asses the similarity between a given action concept and an entity concept. However in order to asses the interoperability between two different ontologies we need to use the notion of virtual evidences explained by [32].

We extend the mechanism to specify virtual evidences by making a fundamental assumption about action concepts in geospatial ontologies. This assumption is based on the work of Kuhn [11] which attempts to make the case for concepts of geospatial actions as primary constituents of geo-ontologies. We assume that our knowledge about increasing complexities of geospatial actions leads to knowledge about the complex environment, which affords such actions, and hence the categorization of geospatial entities and artifacts.

3.3.1 Reasoning Across Ontologies with Common Functions

We now arrive at the basic motivation task of reasoning across ontologies. Since our two texts have differences in the list of geospatial entity concepts (the Highway code contains mention of *Footpath* and *Motorway* whereas the NY driver's manual

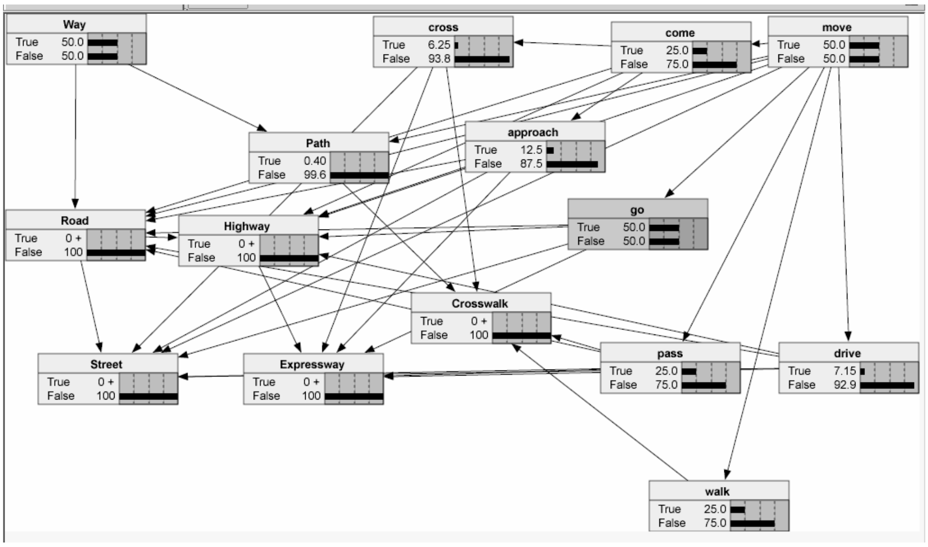


Fig. 3. DAG extracted from the NY Driver Manual text

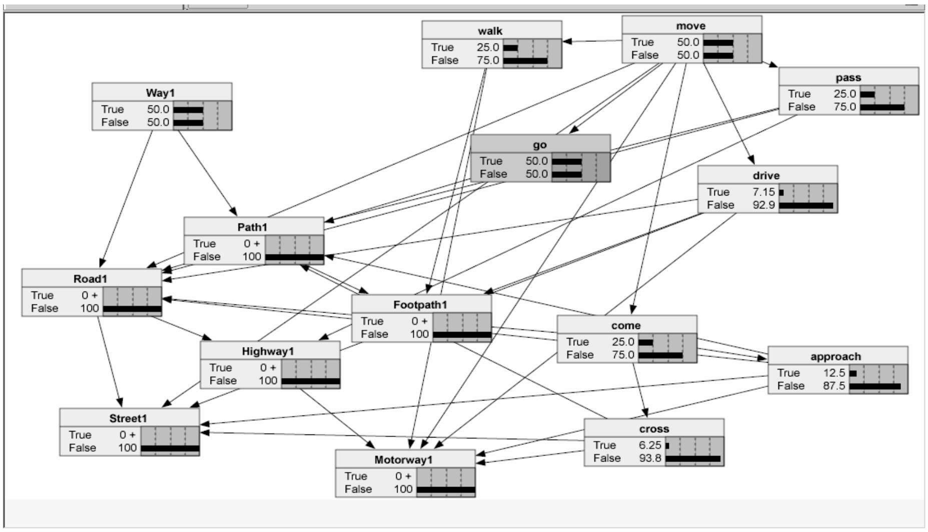


Fig. 4. DAG extracted from the UK Highway code text

mentions *Crosswalk* and *Expressway*, our task is to obtain degree of overlap between these two concepts and the most similar concepts given their linkages with the common function concepts. To do this, we make an assumption that action concepts remain invariant across the ontologies such that the meaning of *walk* or *drive* remain the same (although the meaning of a *Road* and a *Highway* can differ). We create a virtual node for each node of the given ontology in the target ontology based on its conditional probabilities in respect to the action concepts (common to both ontologies). Thereafter we obtain the most similar and most dissimilar concepts based on the approach already used in § 3.2.1.

Table 2 lists these top matches obtained from the two BNs. Further we are also able to obtain the worst matches or most dissimilar concepts as shown in table 3.

Table 2. Most similar and dissimilar concepts of the HWC in the NYDM

HWC Concept	Most similar entity	Most dissimilar entity
Footpath	Path	Expressway
Highway	Way	Street
Motorway	Road	Crosswalk
Path	Path	Expressway
Road	Road	Expressway
Street	Path	Street
Way	Way	Expressway

Table 3. Most similar and dissimilar concepts of the NYDM in the HWC

NYDM Concept	Most similar entity	Most dissimilar entity
Way	Way	Motorway
Street	Way	Street
Road	Road	Street
Path	Path	Motorway
Highway	Path	Street
Expressway	Road	Street
Crosswalk	Path	Motorway

4 Psycholinguistic Verification

We have already stated that a simplistic evaluation of the machine based values of similarity and hence the mapping between concepts of two ontologies is not appropriate. For example the assumption that *Road* in HWC and NYDM mean the same needs to be validated by human experts (this is considered as the ideal case and the ultimate choice while dealing with semantic matchmaking). However the term "expert" is also subjective and so we try to analyse the mappings obtained from different subjects including (i) those who have driven in on of the countries (and hence familiar with entities of one of the ontologies) and (ii) those who have driven in both (and thus familiar with entities from both). We have also maintained a balance of age groups and gender.

This section explains human subjects testing based on the first case study and tries to compare the results of the machine based mappings vis-à-vis human generated ones.

4.1 Human Subjects Tests

Human subject testing was conducted for 20 participants who were native English speakers or with equivalent proficiency. Participants were given two sets of cards, which had names of road network entities from each ontology (the Highway Code and NY Driver's Manual). They were asked to arrange the cards such that the entity from the first set they believed was the most similar to one from the other set, remained closest. After this task was completed, they were asked to flip the cards and read the sections of the texts relevant to the respective entities, which occurred in the corresponding traffic code texts. These sections provided information about the different actions that were permissible on that particular road network entity.

The mappings generated before and after flipping the cards (and hence before and after the knowledge about entity functions was available) were recorded and analyzed. The tests took not more than 20 minutes and were administered with no interference once the initial instructions were given. All 20 participants volunteered willingly and were debriefed at the end of the tests.

The raw data obtained from before and after the flipping of cards were arranged in a two spreadsheets and the differences were recorded in another. One-way ANOVA tests were applied between the values obtained from each human subject to analyze the variance. They show that all subjects have similar matches and the variations are relatively lower for the mappings before the flipping of cards and slightly higher for mappings after the flipping the corresoponding P values from the ANOVA tests are 0. 99 and 0.97. The P value for difference between the two being lower (=0.36) is significant enough to state that the changes between mappings before and after flipping the cards, for all subjects, was similar across the human subjects.

4.2 Analysis

Table 5 below summarizes some of the mappings generated from the human subject tests. We note that most dissimilar mappings are not reported here for sake of simplicity. We note that other than the cases of *Street* and *Motorway* most mappings also appear in machine-based mappings.

Interestingly, the HWC Concept *Street* appears as the most dissimilar concept in the machine based mappings, which is the biggest disparity between the two mappings. However, rest of the mappings is seen to be comparable. The human mappings get closer to the machine-base mappings after the subjects were informed of the functional properties. For example, the confidence values for the mapping between Crosswalk and Footpath decreased after the human subjects were informed about functions of the entities and Path became the most similar concept.

It is also important to note that the covariance of the mapping values in respect to age and gender was found to be insignificant. The variance of mappings produced by subjects who have driven in both countries was found to be slightly lower than those who have driven only in one but this was fairly insignificant.

Table 5. Human generated mappings. Most similar and dissimilar concepts of the HWC in the NYDM (a) before reading the texts about entity functions

NYDM Concept	*Most similar entity*	*Values (0 to 3)*
Way	Way	2.7
Street	Street	2.7
Road	Road	2.65
Path	Path	2.5
Highway	Highway	0.875
Expressway	Motorway	2.55
Crosswalk	Footpath	0.95

(b) after reading the texts about entity functions

NYDM Concept	*Most similar entity*	*Value (0 to 3)*
Way	Way	1.8
Street	Street	2.85
Road	Road	2.95
Path	Path	1.2
Highway	Road	1.25
Expressway	Motorway	2.5
Crosswalk	Path	1.0

We have already discussed that there is a close resemblance in the machine based mappings and the human based mappings although they are not identical. It is possible to report precision and recall of the mappings in terms of false positives (when a true match is overlooked) and false negatives (when a incorrect match is reported), using a Unique Name Assumption (UNA which, assumes that entities which have same names in both ontologies are the same entities). This is not a good evaluation of the performance of the machine based mapping because naming heterogeneity is abundant in most cases. This is also evident from the results of our human subject tests.

Graph 2 compares the precision and recall values based on the UNA and on the mappings produced by the human subject tests. The recall value remains the same (mainly due to the mismatch of the entity *Street* in the machine-based mappings). However recall has been shown to improve. Also as stated earlier, the comparison is more favourable with human mappings when functional knowledge is established. The confidence values are also similar (we avoid these details for lack of space)

5 Applications

Our approach to geospatial ontologies enables practical usage of ontologies and ontology based reasoning to some well-known problems. These include (i) problems of finding most similar and most disimilar concepts within an ontology [33] , (ii) problems of finding most similar and most disimilar concepts across ontologies [34] and

(iii) quanitification of the difference between two given ontologies based on the naming hetrogenities of the entity concepts.

5.1 Concept Mapping Within and Across Ontologies

Mapping or infering most similar concept within an given ontology has been attempted in many contexts [33]. Similarity measurement has been a focus of recent research in the area of semantics of geospatial data [34, 35]. However we need to point out that similariy assesment has been beyond the scope of conventional ontology reasoning. Our approach allows the use of such inferences not only in the context of a given geospatial ontology but also across ontologies. Such mappings have important implications for ontology based mapping of geospatial databases and services [4].

5.2 Interoperability Between Ontologies

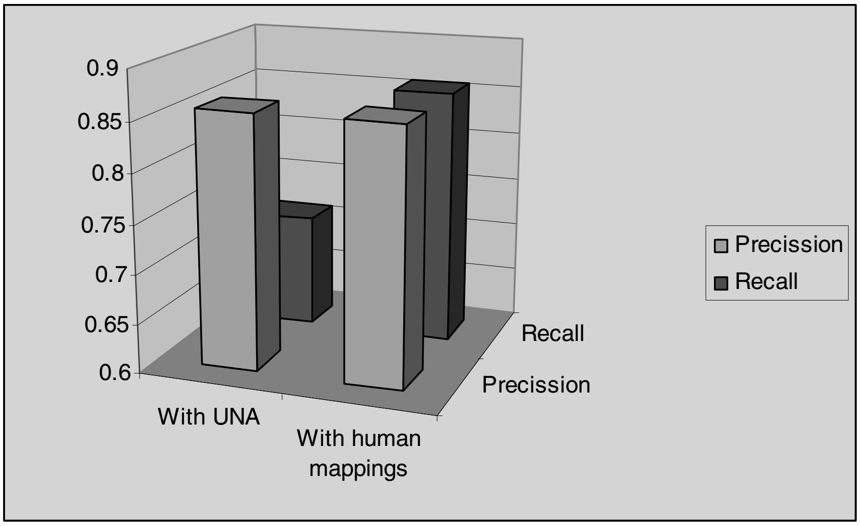
Based on the mappings seen above we not only can ascertain the similar concepts and the dissimilarities but also quantify the value by which the two ontologies match. Interoperability between two geontologies has been defined by Fonseca *et al* as the degree of similarity matches between tuples of concepts of the given ontologies [39]. In our context this can be obtained by the precision and recall values of the terms that are common to both ontologies. It is possible to hypothesize that due to the errors generated by use of wrong affordance values and heuristics used for the same, the true value of interoperability may not be found. For example, in table 2 and 3 the terms *Street* occurs as most dissimilar concept to the corresponding occurrence of *Street* in the other ontology. But even if we assume uniform distribution of errors and a cancellation effect there in, it is possible to report the degree of matches as a measure of the level of interoperability. We can choose to derive this, based on the F value derived from the harmonic mean of precision and recall [28]. This value

$$F = 2 * P * R / (P + R) \qquad \text{where P=Precision and R= Recall} \qquad (2)$$

The measured value in this case results to be 0.779. A value equal to 1 indicated a total match or complete interoperability and a value of 0 is the worst-case scenario. To verify if this was the case in respect to how humans conduct mappings across systems.

We also conducted human subjects testing and revised our estimates of the interoperability between the two ontologies. We use the F values to estimate our interoperability measure, as before. In this case we consider the human generated mappings to obtain the values of precision and recall. There are two sets of values available, (i) for the mappings obtained before the humans subjects were allowed to read the geospatial actions related to the entity by flipping the card and (ii) after flipping and reading the relevant texts about actions.

The computed value of both precision and recall remain at 100% before the flipping of cards and 80% after the flipping. This resulted in interoperability value = 1 in the first case dropping down to 0.8. Incidentally, the value of the later is closer to the machine generated value of interoperability.



Graph 2. Graph comparing evaluations of machine-based mapping in the (i) absence and (ii) presence of human mapping values

6 Conclusions and Future Work

We have reported heuristics based mechanism to specify affordance-based linkages between entitity concepts and action concepts with a geosptial ontology. Stochaistic reasoning upon such ontologies allow inference similarity values and utilizes a simple assumption of invariance of geospatial action concepts. We have deomonstrated the efficiency of our approach based on efficiences of the machine-based mappings.

One important aspect of our work in treating actions as invariant is the philosophical perspective that the meanings of geospatial entities are dependent on the use they have for humans. Although this assumption is a fairly significant one, in the geospatial domain, it requires further validation and acceptance than it currently has. It finds relevance in the theories of human geography, which assigns meanings to a place based on its uses to human society, and the activities that it can afford [37]. We also hypothesize that the framework can be used in respect to any other set of invariant set of concepts that may be common to both ontologies, such as top level ontologies like DOLCE [38].

6.1 Future Work

This is only a first step towards linking ontologies of actions and entities and requires further efforts, which will help to build translationary mechanisms across ontologies and hence enable the applications discussed in Section 4. Our experience has shown that there exist many themes for future work. These include

- Extraction of affordance as outlined in this paper is restricted to text analysis, which can be substituted with field based studies and values obtained from human subjects testing as reported in literature [40].
- Inclusion of Disjoint, Equivalent, Intersection and Union relations: For simplification of our case study these relations were avoided although these relations can be easily determined from WordNet during text analysis. Using such relations in future will require use of some iterative algorithm such as DIPFP in order to enforce truth conditions of the LNodes in BayesOWL[27].
- Testing on industrial scale: Finally this experiment, although at a prototype scale, aims to provide similarities across ontologies, which are developed at an industrial scale. It would be important to evaluate the full-scale deployment of such a methodology to large scales geospatial ontologies.

Acknowledgements

TBD.

References

1. Gruber, T.R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In: Guarino, N., Poli, R. (eds.) Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, Dordrecht (1993)
2. Kashyap, V., Sheth, A.: Semanti Heterogenity in Global Information Systems: The Role of Metadata, Context and Ontologies. In: Papazoglou, M., Schlageter, G. (eds.) Cooperative Information Systems: Current Trends and Directions, Academic Press, London (1998)
3. Fonseca, F.T., et al.: Using Ontologies for Integrated Geographic Information Systems. Transactions in GIS 6(3) (2002)
4. Kuhn, W.: Why, of What, and How? Journal on Data Semantics III (2005)
5. Agarwal, P.: Ontological considerations in GIScience. Int. Journal of Geographical Information Science 19(5), 501–536 (2005)
6. Kuhn, W.: Ontologies in Support of Activities in Geographic Space. International Journal of Geographical Information Science 15(7), 613–631 (2001)
7. Bishr, Y., et al.: Probing the Concept of Information Communities-a First Step Toward Semantic Interoperability. In: Goodchild, M.F., et al. (eds.) Interoperating Geographic Information Systems, pp. 55–71. Kluwer Academic Publishers, Boston (1997)
8. Bittner, T., Frank, A.: On the design of formal theories of geographic space. Journal of Geographical Systems 1(3), 237–275 (1999)
9. Raubal, M., Kuhn, W.: Ontology-Based Task Simulation. Spatial Cognition and Computation 4(1), 15–37 (2004)
10. Rüther, C., Kuhn, W., Bishr, Y.: An algebraic description of a common ontology for ATKIS and GDF (2000)
11. Klien, E., Probst, F.: Requirements for Geospatial Ontology Engineering. In: AGILE 2005. Conference on Geographic Information Science, Estoril, Portugal (2005)
12. Gómez-Pérez, A., Férnandez-López, M., Corcho, O.: Ontological Engineering
13. Ding, Z., Peng, Y., Pan, R.: BayesOWL: Uncertainty Modelling in Semantic Web Ontologies. In: Soft Computing in Ontologies and Semantic Web, Springer, Heidelberg (2005)

14. Sen, S.: Linking hierarchies of entities and their functions in geospatial ontologies. Journal of Geomatics 1(1) (2007)
15. Grenon, P., Smith, B.: SNAP and SPAN: Towards Dynamic Spatial Ontology. Spatial Cognition and Computation 4(1), 69–104 (2004)
16. Camara, G., Monteiro, A.M.V., et al.: Action-Driven Ontologies of the Geographical Space. In: GIScience 2000, Savannah, GA, AAG (2000)
17. Timpf, S.: Geographic Task Models for geographic information processing. In: Duckham, M., Worboys, M.F. (eds.) Meeting on Fundamental Questions in Geographic Information Science, pp. 217–229 (2001)
18. Worboys, M.: Event-oriented approaches to geographic phenomena. Int. Journal of Geographical Information Science 19(1), 1–28 (2001)
19. Johansson, I.: Functions, Function Concepts, and Scales. The Monist 87(1), 96–114 (2004)
20. Kitamura, Y., Koji, Y., et al.: An Ontological Model of Device Function and Its Deployment for Engineering Knowledge Sharing. In: First Workshop FOMI 2005 - Formal Ontologies Meet Industry, Castelnuovo del Garda (VR), Italy (2005)
21. Koller, D., Levy, A., Pfeffer, A.: P-CLASSIC: A Tractable Probabilistic Description Logic. In: Proc. of AAAI 1997, pp. 390–397 (1997)
22. Straccia, U.: A fuzzy description logic. In: AAAI 1998. Proc. of the 15th Nat.Conf. on Artificial Intelligence, Madison, USA, pp. 594–599 (1998)
23. Stuckenschmidt, H., Visser, U.: Semantic Translation based on Approximate Reclassification. In: KR 2000. Proc. of the Workshop Semantic Approximation, Granularity and Vagueness (2000)
24. Doan, A., Madhavan, J., Halevy, A.: Ontology Matching: A Machine Learning Approach. In: Staab, S., Struder, R. (eds.) Handbook on Ontologies in Information Systems, pp. 397–416. Springer, Heidelberg (2004)
25. Costa, P.C.G, Laskey, K.B.: PR-OWL: A Framework for Probabilistic Ontologies. In: FOIS 2006. International Conference on Formal Ontology in Information Systems, IOS Press, Amsterdam (2006)
26. Patwardhan, S., Pedersen, T.: Using WordNet Based Context Vectors to Estimate the Semantic Relatedness of Concepts. In: EACL 2006. Workshop Making Sense of Sense - Bringing Computational Linguistics and Psycholinguistics Together, Trento, Italy (2006)
27. Peng, Y., Ding, Z.: Modifying Bayesian Networks by Probability Constraints. In: UAI 2005. 21st Conference on Uncertainty in Artificial Intelligence, Edinburgh, Scotland (2005)
28. Rijsbergen, V.: Information Retrieval, 2nd edn. Butterworths, London (1979)
29. Kuhn, W.: Ontologies from Texts. In: GIScience 2000, Savannah, Georgia, USA, University of California Regents (2001)
30. Fellbaum, C. (ed.): WordNet - An Electronic Lexical Database. MIT Press, Cambridge (1999)
31. Fellbaum, C.: On the Semantics of Troponymy. In: Green, R., Bean, C., Myaeng, S. (eds.) The Semantics of Relationships: An Interdisciplinary Perspective, pp. 23–24. Kluwer, Dordrecht (2002)
32. Pan, R., et al.: A Bayesian Network Approach to Ontology Mapping. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, Springer, Heidelberg (2005)
33. Cross, V.: Fuzzy semantic distance measures between ontological concepts. IEEE Annual Meeting of the Fuzzy Information , pp.635–640

34. Rodríguez, A., Egenhofer, M.: Determining Semantic Similarity Among Entity Classes from Different Ontologies. IEEE Transactions on Knowledge and Data Engineering 15(2), 442–456 (2003)
35. Janowicz, K., Raubal, M.: Affordance-Based Similarity Measurement for Entity Types. In: Winter, S., Duckham, M., Kulik, L., Kuipers, B. (eds.) COSIT 2007. LNCS, vol. 4736, Springer, Heidelberg (2007)
36. Schwering, A.: Hybrid Model for Semantic Similarity Measurement. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3761, Springer, Heidelberg (2005)
37. Tuan, Y.-F.: Space and Place: The Perspective of Experience. University of Minnesota Press, Minneapolis (1977)
38. Masolo, C., et al.: WonderWeb Deliverable D17. The WonderWeb Library of Foundational Ontologies and the DOLCE ontology (2002)
39. Fonseca, F.: A Framework for Measuring the Interoperability of Geo-Ontologies. Spatial Cognition and Computation 6, 307–329 (2006)
40. Raubal, M.: Formalizing Conceptual Spaces. In: Varzi, A., Vieu, L. (eds.) FOIS 2004. Proceedings of the Third International Conference. Frontiers in Artificial Intelligence and Applications, vol. 114, pp. 153–164. IOS Press, Amsterdam (2004)

OntoCase - A Pattern-Based Ontology Construction Approach

Eva Blomqvist

Jönköping University, Jönköping, Sweden
blev@jth.hj.se

Abstract. As the technologies facilitating the Semantic Web become more and more mature they are also adopted by the business world. When developing semantic applications, constructing the underlying ontologies is a crucial part. Construction of enterprise ontologies need to be semi-automatic in order to reduce the effort required and the need for expert ontology engineers. Another important issue is to introduce knowledge reuse in the ontology construction process. By basing our semi-automatic method on the principles of case-based reasoning we envision a novel semi-automatic ontology construction process. The approach is based on automatic selection and application of patterns but also includes ontology evaluation and revision, as well as pattern candidate discovery. The development of OntoCase is still ongoing work, in this paper we report mainly on the initial realisation and first experiments concerning the retrieval and reuse phases.

1 Introduction

Today companies struggle with a large information overflow. The size of company intranets is soon comparable to the size of the entire web some years ago. When developing semantic applications for enterprises, constructing the underlying enterprise ontologies is a crucial part. Ontology engineering has for a long time been considered a manual task, but it is a very resource demanding and tedious process, so therefore many methods proposed today are semi-automatic. Using semi-automatic approaches reduce both the total construction effort and the need for certain ontology engineering expertise. The field of semi-automatic ontology construction is still a relatively new research field and most methods leave many difficult tasks up to the user. This is one thing that our research aims to improve, through further automation of the process in combination with increased output quality.

An important issue is also to introduce knowledge reuse in the ontology construction process. For instance, in the business domain there are many similarities between companies, e.g. how they organise their business processes and their information. Common practises should be exploited, as well as drawing on best practises in ontology engineering to correctly model the features. One way of doing this would be through ontology patterns. Case-based reasoning (CBR) is a methodology also based on the idea of reuse. By combining the CBR viewpoint with the use of patterns, the OntoCase approach intends to apply a novel hybrid view of semi-automatic ontology construction, both encompassing patterns and reuse of partial solutions.

The following sections briefly introduce some background and definitions together with related work. In Section 3 we focus on the research questions that the OntoCase approach is based on and in Section 4 two initial experiments are described, which additionally motivates the use of a CBR viewpoint. Next, in Section 5 the phases of the OntoCase approach are described. Finally, the paper is concluded with an outlook and some preliminary conclusions in Section 6.

2 Background

This section describes some background relevant to the rest of this paper, including definitions and related work in ontology engineering and ontology patterns.

2.1 Ontologies

In our research we adopt the commonly used ontology definition from [1], stating that an ontology is a formal explicit specification of a shared conceptualisation. In our view, this means that an ontology generally contains concepts, a taxonomy, general relations, and possibly other axioms. We do not, at this stage, restrict our research to one specific ontology representation formalism, but the OntoCase approach assumes the possibility to reduce the ontology to a semantic network-like representation (a directed labelled graph). We realise that such a reduction is not trivial and might result in loss of information, but the details of this problem is considered outside the scope of this paper.

One way of describing ontologies is to view the levels of abstraction in an ontology, as defined in [2]. In this context our research focuses mainly on domain and application ontologies within enterprises. Another categorisation of ontologies is to classify them by their intended use, and thus we focus on ontologies for structuring and retrieval of information with respect to one enterprise. This kind of ontology we denote *enterprise application ontology* throughout this paper. An enterprise ontology (see [3]) generally intends to cover all the knowledge of an enterprise, and is classically acquired during an enterprise modelling process.

2.2 Ontology Patterns

A general view of patterns is the one commonly used in pattern mining, where the aim is to find regularities in some set of objects, like recurring sets of ontology primitives in a set of ontologies. We restrict this view slightly to only consider connected sets of ontology concepts (*connected* referring to the ontology as a graph). Finding such patterns is in our case, as we shall see later, comparable to the CBR idea of retaining partial solutions. A different view is when patterns denote predefined templates for constructing new solutions. This is a more consensual view, where patterns aim to encode the best practises of some community or domain. These patterns are usually more abstract and perhaps not formally represented, while in the pattern mining view a pattern can be any reoccurring partial solution. In our research we aim to exploit a combination of both these views. With the single term *pattern* we will throughout this paper denote patterns in a very general sense, encompassing both views described above. More specific terms will also be used, then referring to either view described above, like *ontology design patterns* described in the following paragraph that refers to patterns conforming to the best practises view of patterns.

When discussing patterns in the second sense mentioned above, patterns as encoding best practises, we are in the OntoCase approach mainly interested in patterns on the design and architecture level (see [4] for a classification). We describe the notion of ontology design patterns generally as self-contained ontology templates for constructing some ontology component, each consisting of a well-defined set of ontology primitives. Ontology design patterns belong to the tradition of consensual best practises and can be compared to for example software design patterns. Related work on ontology design patterns mainly focus on manual approaches, as the content patterns described in [6], and language specific modelling idioms as described by [7], although recent research, as in [8], is aiming towards automatic use of content patterns for example. An ontology architecture pattern is a set of constraints on the way the ontology as a whole can be composed, analogous to the notion of architecture in for example software engineering. So far no architecture patterns have been constructed, thereby to encompass architecture patterns in the practical realisation of OntoCase is left to future work.

2.3 Ontology Engineering

Research in the area of ontology engineering, for the Semantic Web and related applications, is progressing fast. Recent developments involve semi-automatic ontology construction, sometimes denoted ontology learning (OL). Most of the semi-automatic approaches focus on techniques for text analysis in order to extract mainly suggestions for concepts and relations from a text corpus input. None of the existing semi-automatic approaches exploit ontology design or architecture patterns, to the best of our knowledge. Recent systems can be found in [9], [10], [11] and [12], mainly based on a variety of text analysis techniques. The latest additions involve algorithms for named relations extraction as described in [13] and extraction of disjointness axioms as in [14] for example. Additional ongoing research efforts, like [8], aim at exploring the possibilities of semi-automatic pattern usage and networked ontologies.

An important requirement, as we shall see later, is multi-word term extraction, since specific concepts rarely can be described using only one word. Relation extraction can be done both using lexico-syntactic patterns and other techniques, like collocation analysis. Usually each extracted primitive is associated with a confidence value, representing some kind of certainty that the primitive is correctly extracted and relevant. The confidence value of a term or relation is commonly determined using different corpus statistical measures. A general problem with the use of a text corpus as input is that not all information is explicitly stated (as discussed in detail in [15]). Some approaches use additional knowledge sources, dictionaries (like WordNet [16]) or the web (as in [12]), for supporting the ontology construction. Another way to reduce the impact of this problem is to use ontology patterns, as we attempt in the OntoCase approach.

Ontology matching is a field that partly aims to assist ontology reuse in ontology engineering. Surveys of ontology matching algorithms and techniques can be found in [17] and the more recent [18]. The OntoCase method is not a new ontology matching approach as such, still we intend to use many of the same techniques when for example matching the input to the patterns and combining the patterns. One widely used technique is string matching (see [19] and [20] for surveys and comparisons of string matching metrics). Our approach is also related to approaches for searching and ranking ontologies (assisting ontology reuse by finding relevant ontologies to integrate) like the ontology ranking scheme described in [21]. Most ontology search engines though, are based on a very simple query (set of keywords) that is evaluated against the available set of ontologies. Although the actual ranking might be more elaborate, like in later versions of the Swoogle engine (see [22]). In contrast we would like to compare two more rich structures, and then rank one with respect to the other.

2.4 Case-Based Reasoning

Case-based Reasoning (CBR) is, according to for example [23], aiming to use previous experiences to solve new problems. It is worth noting that this is a quite similar idea compared to patterns, both aim at reuse of knowledge and experiences. The CBR process is generally depicted as a cycle of four phases: retrieve, reuse, revise and retain. The retrieval phase constitutes the process of deriving a suitable representation of the new case (the process input) and then compare this representation to the stored previous cases in the case base. Next, the best match (or several matches) from the case base are retrieved and used as input for the second phase. In the reuse phase the retrieved case(s) are reused and adapted to fit the current representation of the new case (the input); an initial solution is formed. In the third phase, revision, the initial solution is evaluated and revised until it is sufficient as output of the process (or until the process is deemed to fail, if the result is still not appropriate). Finally, the fourth phase is a learning step when the new solution, or parts of it, can be retained and stored in the case base for future reuse. Also feedback from the solution construction process can be retained and stored.

A specific branch of CBR is Textual CBR (TCBR) that focuses on approaches using natural language texts, and their representations, in the CBR process. In [24] four research questions of this area are stated. One question is concerned with how to get from a textual representation of a case to a more structured representation, and another question is concerned with automating the approaches. Our method incorporates both these research questions, in the specific setting of semi-automatic enterprise ontology construction, although the first question is mainly addressed through the use of existing OL approaches. The second question, concerning further automation of the TCBR process is highly relevant with respect to the OntoCase method. Some additional recent developments in the CBR field also use "soft" computation techniques (see [25]) to represent uncertain information and human-style decision making. The OntoCase approach also incorporates uncertainty throughout the process, due to the imprecise nature of both automatic text analysis, matching, and aggregation of these results.

3 Research Questions

In our research we have stated a number of long term goals that address some of the open issues of the OL field. The OntoCase approach is mainly based on research addressing the following research questions:

- How can the effort of constructing an enterprise application ontology be reduced?
  - How can the methodology of CBR improve semi-automatic ontology construction?
- How can knowledge and experience be reused in ontology engineering?
  - How can ontology patterns improve semi-automatic ontology construction?

Based on the research questions the following hypotheses have been proposed:

- Automation reduces the total construction effort.
  - CBR gives a framework for further automation of the ontology construction process compared to related semi-automatic approaches that exist today.
- Domain knowledge and engineering experiences can be reused through patterns.
  - Using CBR together with ontology patterns can improve the quality of the generated ontologies compared to existing semi-automatic approaches.

To test the hypotheses, the OntoCase approach is being developed based on the methodology of CBR and the notion of ontology patterns, and will later be evaluated and compared mainly to related OL approaches.

4 Initial Experiments

Two initial experiments have been performed. The first one explores the boundary of the OntoCase approach towards existing OL approaches, and concludes that OntoCase should incorporate state-of-the-art OL algorithms for text processing as a pre-processing step before applying our pattern-based method. To explain this experiment an example is described in the next section. In addition a partial version of the OntoCase approach was tested and evaluated in a real-world project, for a detailed description of this see [?]. In this second experiment only the first two phases of the cycle were implemented, mainly restricting the method to a pattern selection and combination approach. The analysis of this experiment motivates the case-based reasoning viewpoint and the additional improvements of the method that have been made since this experiment. The second experiment is further described in Section 4.2.

4.1 Text Processing for Ontology Learning

A small evaluation has been performed comparing one of the recent OL systems freely available, to more basic text analysis techniques like tokenisation, morphological normalisation and stop word removal etc. This was done to exemplify why pattern-based methods like OntoCase needs to rely on recent OL systems, rather than standard text processing tools. For this example the collection of concept extraction algorithms in the research prototype called Text2Onto (see [9]) was chosen, both due to that it incorporates a large number of different algorithms and the fact that it is freely available for download. The example is not intended as a quality measure of the particular approach, or approaches like [9] in general, it only aims to point out that certain features like multi-word term extraction (through algorithms as in [26], also used in Text2Onto) are needed for OL, and that multi-strategy approaches seem to give more useful results than classical text processing. This is also our motivation why we intend to build on existing OL systems (even though they are mostly research prototypes) for input processing, instead of using standard language processing components.

In the example a text corpus was first manually analysed by an ontology engineer, who picked the terms from the text he most likely would use as labels of concepts (and label synonyms) when building an ontology to represent the texts. This manually constructed term list was used as a ”gold standard”, when comparing the result of Text2Onto to different combinations of basic techniques (tokenisation, weak stemming, and stop-word removal etc.) in standard implementations (mainly the GATE ANNIE framework, as described in [27]).

Analysis. Some example results are shown in Table 1. The experiment covered several additional combinations, but the ones presented below are sufficient to illustrate the general issues. The low level of recall using standard components (see experiment 1-3) is mainly due to the fact that the classic techniques do not consider multi-word terms, while this can be done through more elaborate algorithms in tools like Text2Onto. Additionally it is obvious that the OL approach (see experiment 4) performs a much more effective and tailored filtering to also increase the precision. This is mainly done through specific term relevance measures and lexico-syntactic patterns applied on the parsed text.

In general, existing OL approaches do not only transform terms and linguistic annotations (determined through classic natural language processing techniques) to an ontology representation, but additionally use algorithms specifically tailored towards discovering ontology primitives and filtering out irrelevant information. A problem with using existing OL systems (many of which are still experimental prototypes) as a basis for OntoCase could be to introduce a bias, and possibly filter out relevant information. Although our small experiment indicates that the extracted primitives of the OL system also better agree with human intuition (the decisions of a human ontology engineer) than simply presenting every word from a text, we recognise this risk since the recall is still below 50% compared to manual extraction. Still, using existing OL algorithms clearly improve a lot on using classical text analysis techniques (which is the only alternative in terms of automatic processing) for our specific case.

Table 1. Precision and recall of text analysis for concept discovery

Experiment	Method	Precision	Recall
1	ANNIE tokenisation	14%	33%
2	ANNIE token. + weak stemming	16%	37%
3	ANNIE token. + weak stemming + stop words rem.	18%	37%
4	Text2Onto	52%	48%

4.2 Pattern Selection and Combination

The second experiment was performed in the context of the SEMCO project. SEMCO aims at introducing semantic technologies into the development process of software-intensive electronic systems. The experiment also included a manual construction of an ontology with the same scope, completely separate from the automatic experiment, in order to later analyse similarities and differences between the ontologies. The process steps included were focused on extraction of terms and relations from the input text corpus (using existing tools), comparing this to the concepts and relations of the patterns, and computing a similarity score for each pattern. Patterns above a user-defined threshold were selected and combined through a naïve approach, discarding parts that did not have appropriate support. For details concerning the experiment setup and pattern examples see [**?**]. Both more general ontology design patterns and other reusable components were used as *patterns* in this experiment, conforming to the more general notion of patterns as discussed in Section 2.2.

Analysis. When compared to the analysed input (details on the evaluation are described in [28]), the automatically constructed ontology covered only 34% of the terms. The reason is partly a small pattern catalogue, but when compared to the manually constructed ontology it was mainly quite specific terms that were missing. A selection process is not enough to cover the scope, since patterns are too abstract compared to a text corpus. Additionally, some abstract information was missing since this is not explicit in the texts. Finally, the semi-automatically constructed ontology also had some nice features when compared to the manually constructed ontology, for example a larger number of relations connecting the concepts and to some extent a better structure. This analysis lead us to believe that the pattern selection and combination approach is not enough to substantially improve on current OL approaches. Ways to introduce a general structure and abstract concepts are needed, as well as ways to incorporate the most specific terms within the context and the structure of the patterns.

5 OntoCase

The following sections describe the general outline of the OntoCase approach, and details on the retrieval and reuse phases that are currently being implemented as a prototype system. Also some notes on future work are presented.

5.1 OntoCase Outline

One of the main elements of a CBR approach is the case base and its content. In the OntoCase approach (illustrated at the centre of Figure 1), the case base corresponds to a pattern catalogue (pattern base), containing both ontology design patterns, architecture patterns (tentatively, although no such patterns have been constructed yet) and patterns in the broader sense of reoccurring reusable assets as discussed in Section 2.2. The patterns on design level are constructed for automatic use and are therefore small self-contained ontologies described in some ontology representation language (examples presented in [**?**]).

The first OntoCase phase corresponds to case retrieval and constitutes the process of analysing the input (text corpus) and matching it to the pattern base, to select appropriate patterns. The second phase, case reuse, constitutes the process of reusing retrieved patterns and constructing a first version of the ontology. The third phase concerns revision of the ontology to improve the fit and the ontology quality. The final phase includes the discovery of new patterns as well as storing pattern feedback. In Figure 1 the OntoCase process is described at the centre of the figure, and surrounding it an example instantiation of the complete process is illustrated. The process starts from the top of the figure (with the input provided by the user) and proceeds clockwise in the illustration. Certainly the process is not limited to being applied in a linear fashion, the intention is to be able to apply each phase independently or iterate if necessary, depending on available input data. Each phase in itself may also contain iterations, but for the sake of simplicity we will here describe the process step by step.

In the OntoCase approach there is an uncertainty inherent in all the described steps. Each primitive found in the analysis of the input (mainly terms and relations) has a certain degree of confidence associated with it, and so have the pattern primitives. The pattern confidences are based on a set of factors, where one factor is the ”nature” of the pattern, i.e. whether it is a consensual ontology design pattern, if it was manually constructed (representing best practises and domain consensus), or if it is a pattern retained from one or several solutions. The pattern primitives match input primitives only to a certain extent, and the levels of confidence are transferred onto the constructed ontology when it is built.

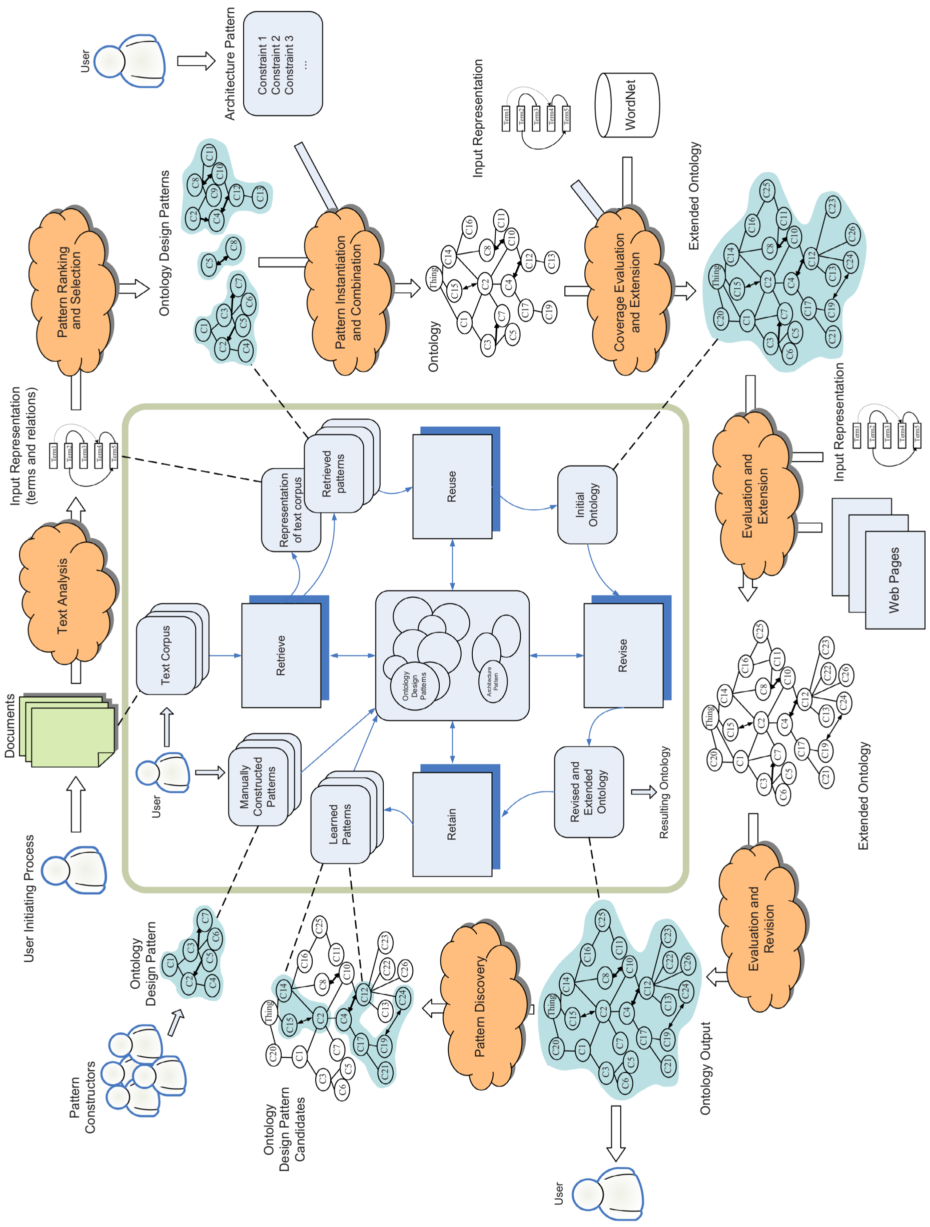


Fig. 1. The OntoCase approach

To store the ontology using a more standard ontology representation formalism, thresholds can be set for acceptable confidence levels or the ontology primitives can be validated by a user.

When comparing the complete OntoCase cycle to the initial approach used in the experiment described previously, the experiment covered two of the four phases. The analysis of the input text corpus and selection of patterns were in the experiment realised using existing text processing and string matching algorithms, and the combination used a naïve implementation. Compared to the conducted experiment the phases have been considerably improved in the current OntoCase approach. The description of the retrieval and reuse phase below shows actual solutions while the discussion of the revision and retain phases describes intended solutions and future work, since OntoCase is still ongoing research.

5.2 Pattern Base

The pattern base contains two distinct kinds of patterns, patterns on the design and on the architecture level. Among the patterns on design level also the consensual and abstract ontology design patterns, as described in section 2.2, are represented as small ontologies. Intuitively, ontology design patterns need a certain level of abstraction to be reusable in several cases, just as patterns in other areas. Still, ontology design patterns for automatic use need to be specific enough to be matched against the input representation, and they need to be formally represented. All the patterns on design level are thereby small ontologies, with the restriction that the graph representation of the pattern must be connected. Examples of slightly simplified versions of two patterns intended for enterprise application ontology construction that were used in the initial construction experiment described in section 4.2 are illustrated in Figure 2 and 3. Whereas the *positions* pattern clearly qualify as a consensual ontology design pattern, the *communication event* pattern has so far been treated as a reoccurring solution rather than an ontology design pattern, since consensual acceptance and validation of the pattern is still uncertain.

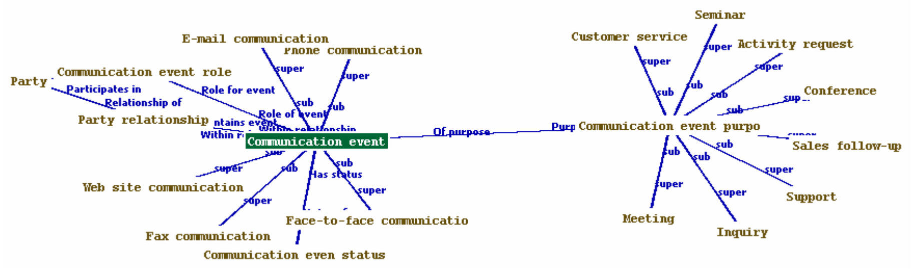


Fig. 2. A pattern covering communication event concepts

The first pattern, illustrated in Figure 2, contains concepts connected to communication within and between companies. A communication event has a certain purpose in the organisation and a specific role for a specific party involved in

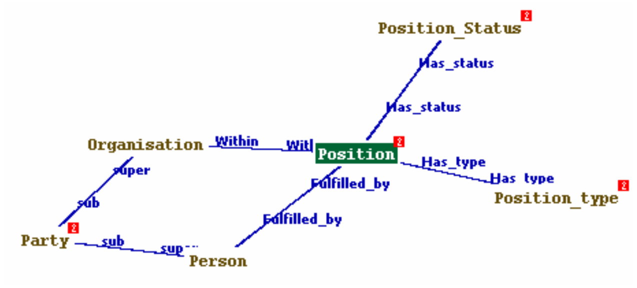


Fig. 3. A pattern covering positions

the communication. This is an example of a relatively detailed pattern, encoding domain knowledge of the business domain (but not necessarily any specific business area). Both patterns presented as examples originate in data model patterns for enterprise database construction, but were slightly modified (replacing relational model specifics with ontology modelling best-practises, see [?]). The transformation was made manually by an ontology engineer. The second pattern example, illustrated in Figure 3, represents a more abstract pattern, better qualifying as an ontology design pattern, dealing with positions within an organisation and their fulfilment by specific persons. This is a simplified version of a pattern similarly inspired by data model patterns.

In our hybrid method, ontology design patterns are constructed manually although candidate patterns may be discovered in produced solutions. The manual pattern construction process involves reaching a domain consensus and encoding of best-practises in ontology modelling. The OntoCase method does not focus on this manual process, but assumes the existence of a pattern base. In case there is a lack of best-practises, or consensus, also the possibility of adapting modelling patterns from other areas (like database construction and problem solving methods) could be used as previously described in [?]. In addition to the approved ontology design patterns, possibly useful variants and specialisations of these design patterns might be stored as reusable assets. In this way the distinction between mined patterns (as reoccurring solutions retained from constructed ontologies) and consensual ontology design patterns will not be very strict in the OntoCase approach, as a matter of fact in the realised automatic construction process no distinction is made at all. The automatic discovery of pattern candidates from solutions is still future work, as described later.

The notion of architecture patterns is used to ensure an appropriate overall structure of the constructed ontology, and to impose constraints on the ontology construction process. In an abstract sense, an ontology architecture pattern could for example describe a division of the ontology into modules, layers and subject areas. Such a general structure could utilise a top-level enterprise ontology similar to what is described in [3]. In the OntoCase approach an architecture pattern is represented as a set of constraints, both used when composing the ontology from the selected ontology design patterns and also as a domain specification or restriction when searching for patterns in the pattern base. So far no architecture patterns have been constructed, mainly due to the lack of available enterprise ontologies, but the theoretical notion of architecture patterns will still be considered in the OntoCase approach.

The pattern base is currently realised in the form of a database. The ontology design pattern part contains metadata of each pattern, pointers to the actual pattern, and connections between patterns (including connections between possible architecture patterns and design patterns). The metadata is part of the pattern base index and is currently based on the labels of the core concepts of each pattern, a set of manually entered keywords (optional), the domain of the pattern (optional connection to an architecture pattern), and the name of the pattern. Connections to other patterns can currently be of two kinds, either the pattern is a "variant of" another pattern or it is simply "related to" another pattern. The variant relation indicates changed patterns, patterns with a significant overlap, and is also used to connect the consensual design patterns to patterns that are retained solutions. When patterns are retrieved for matching and ranking, based on a database query, also direct variants and related patterns will be retrieved.

5.3 Retrieval

The first step of the retrieval phase involves extracting a representation of the input text corpus. The input could in theory be of different kinds (like HTML pages etc.), but so far we assume plain text files. The extraction involves analysing the input text corpus to extract as much "evidence" relevant to ontological primitives as possible. This analysis is the main focus of many of the existing OL approaches, as stated in section 2.3 and studied through the experiment in section 4.1. Therefore developing new solutions for this is not the focus of our research, merely to evaluate and select existing approaches that fit with the OntoCase approach, but such studies is still outside the scope of this paper.

The second step of the retrieval phase involves comparing the input representation (the list of terms and relations with associated confidence values) to the pattern base and select appropriate patterns. When viewing the matching on an abstract level it is similar to ontology matching, with the addition that both "ontologies" (the input representation and the pattern) are uncertain. On a more detailed level there are also some specific characteristics of the ontologies being matched, namely the patterns are relatively small and well structured while the input representation can be very large and usually quite diverse and sparsely connected. The matching results is a special kind of ontology alignment (used in the next phase for composing the ontology) but for the use in this step only a set of similarity connections (each associated with a value) between primitives of the input representation and the patterns is used to compute the total ranking value. The matching method developed is heavily inspired by recent ontology ranking schemes, like the one described in [21], although our approach draws on the richer structure of the input extracted from the text corpus.

We propose a ranking scheme (presented in more detail in [29]) based on four different factors; concept coverage, relation coverage, density and semantic proximity of matched concepts in the pattern. Each pattern is evaluated against the factors and a single value representing the rank of the pattern can be computed. The pattern selection is made based on choosing patterns in the ranking order and computing the resulting total coverage over the input representation. Patterns are chosen until a threshold is reached, either a sufficient coverage has been obtained or the increase in coverage is no longer significant.

To illustrate this process through an example we use the *positions* pattern presented in Figure 3 and a pattern describing *work effort* and its connection to the realisation of requirements, as illustrated in Figure 4 below. The example input representation is based on a set of randomly selected short passages extracted from the text corpus used in the experiment of Section 4.2 (project plans and process descriptions of a company in the automotive industry). The details of the term extraction is not relevant here, but it results in a term list of 33 terms as shown in Figure 5, which will be used throughout this example.

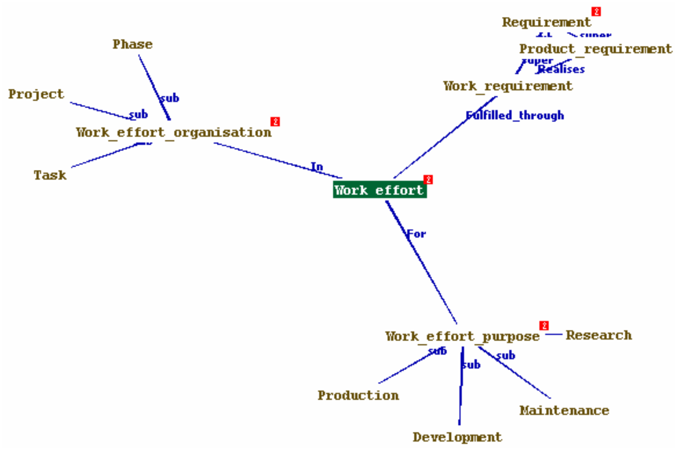


Fig. 4. A pattern covering work efforts and requirements

Details of the ranking process can be found in [29]. For this simple example we only compute the concept coverage and use this as the pattern rank. Concept coverage is based on a direct and an indirect part, the direct part being term and label similarity (determined through string matching), and the indirect part being subsumption coverage (determined for example through the WordNet dictionary and the vertical relations heuristic, see [29]). We apply the Jaccard string similarity measure (for details see for example [19]) with a threshold of 0.5 on the term list and the two patterns. The concept labels of the *positions* pattern in Figure 3 have no matching terms in the term list. On the other hand, the *work effort* pattern from Figure 4 displays several matches, see Table 2.

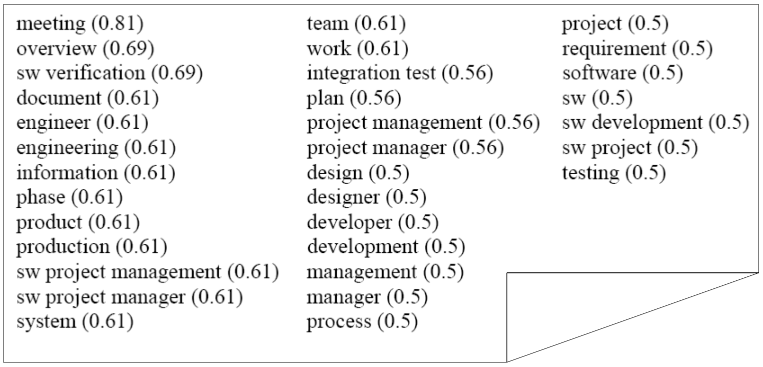

meeting (0.81)
overview (0.69)
sw verification (0.69)
document (0.61)
engineer (0.61)
engineering (0.61)
information (0.61)
phase (0.61)
product (0.61)
production (0.61)
sw project management (0.61)
sw project manager (0.61)
system (0.61)
team (0.61)
work (0.61)
integration test (0.56)
plan (0.56)
project management (0.56)
project manager (0.56)
design (0.5)
designer (0.5)
developer (0.5)
development (0.5)
management (0.5)
manager (0.5)
process (0.5)
project (0.5)
requirement (0.5)
software (0.5)
sw (0.5)
sw development (0.5)
sw project (0.5)
testing (0.5)

Fig. 5. List of extracted terms with associated confidence values

Table 2. String matching scores between labels of the work effort pattern and the extracted terms

Pattern concept label	Extracted term	Similarity
requirement	requirement	1.0
project	project	1.0
phase	phase	1.0
development	development	1.0
production	production	1.0
product requirement	product	0.5
product requirement	requirement	0.5
work requirement	work	0.5
work requirement	requirement	0.5
work effort	work	0.5
project	project management	0.5
project	project manager	0.5
project	sw project	0.5
development	sw development	0.5

Additionally indirect concept coverage measures are applied to discover (with assistance of WordNet and heuristics as mentioned above) that for example the terms *designer*, *engineer*, *developer* and *manager* in the *positions* pattern are concerned with people, thereby indicating that the concept *person* in the pattern might indirectly cover these terms. Similarly, some terms can be found that are possibly covered by the concept of *organisation*. In total this gives the *positions* pattern the aggregated concept coverage value of 0.27. For the *work effort* pattern also some indirectly covered terms can be found and together with the direct coverage (the string matching results) this results in a value of 0.35. In these numbers the uncertainty of the extracted terms is incorporated, as well as the uncertainty of the matches and the fraction of the pattern that is covered (for details see [29]). In the complete method the ranking algorithm would proceed with calculating the relation coverage, as well as density and proximity of the matched concepts, before aggregating these into a rank value.

The important benefits of this approach is the ability to rank abstract patterns, like the *positions* pattern, since the ranking is not only based on simple string matching. Additionally the complete ranking algorithm takes into account extracted and matched relations, and the quality of the patterns in terms of their structure in relation to the matched concepts. For pattern selection the total coverage of the pattern(s) over the input representation should be computed. Selection can then be made based on what the user defines as sufficient coverage, as well as the coverage gain in selecting more patterns.

5.4 Reuse

The reuse phase is concerned with instantiating and combining the patterns into an initial solution ontology. The combination process uses the architecture pattern chosen by the user (if present) to guide the composition, while iterating over all chosen patterns. The default rule is to only include those parts of a pattern that had some match in the input representation. This includes choosing only matched concepts, choosing appropriate labels (and their synonyms), as well as choosing relations. The process is enhanced by a set of heuristics, intended to create a more well-structured ontology. In the example presented in the last section, for the *positions* pattern the default rule would imply to only include the concepts *person* and *organisation* (that covered some input primitives), but additionally a heuristic could be used to include the superconcept *party* if it is desirable to keep the instantiated pattern "connected". Another such heuristic is for example to use the transitive property of taxonomic relations, including more taxonomic relations to keep the ontology connected even if not all intermediate concepts were matched and included.

Also pattern overlap needs to be resolved during the ontology construction. For some patterns explicit "variant of" relations exist in the pattern base, then these can be used to assume an overlap between patterns. Overlap can additionally be handled using heuristics, for example assuming that two concepts represent the same concept if they have the same set of synonyms and no conflicting relations (or other axioms). This is a naïve approach used for the first version of OntoCase that needs further refinement in future work, and should conform to more advanced methods of ontology integration and merging. The confidence values of the pattern primitives and input representation primitives are used together with the matching values to compute new confidence values for the primitives of the resulting ontology.

The evaluation criteria used for the second part of this phase is coverage of the input representation (as when selecting patterns in the previous phase). The second part of the reuse phase is devoted to extending the generated ontology with primitives from the input representation. The primitives not sufficiently covered are ordered according to their confidence values and a connection to the resulting ontology is investigated. This connection might be an extracted relation, but connections might additionally be found in external sources (as already described in the matching step above). Currently WordNet is used as such an external knowledge source. If a connection is discovered the primitive can be added, but the confidence is quite low since WordNet relations not necessarily correspond to ontological relations, and additionally WordNet is a very general source of knowledge. The process continues until a sufficient total coverage is reached or no more primitives can be added with acceptable certainty.

5.5 Future Work - Revise and Retain

For the revision phase, one objective is to compensate the missing background information of input texts, i.e. to really cover the complete domain intended. Some missing parts have already been added with the help of design patterns, but still we can see from the initial experiment that this will probably not be enough. In this case we have to use external sources of information to try and attach primitives to the ontology in a structured way. Our idea for improvement involves using selected (focused) parts of the web to gather more general information. In addition a second step of the revision phase will focus on reduction of redundancy and resolving inconsistencies in the ontology.

In the final phase, retaining patterns, we have mainly been inspired by approaches to ontology modularisation and algorithms for finding strongly connected graph components. Finding coherent parts of the ontology that might constitute suggestions for new patterns can be done by traversing the taxonomy, and through heuristics the candidates can be restricted in their size and structure. We envision some user involvement in this step, validating and possibly generalising the candidates before inclusion in the pattern base. The feedback process for existing patterns, involves the recalculation of confidence values present in the applied patterns, as well as supporting the user in making pattern changes or updates. Purely manual pattern construction methods are considered outside the scope of OntoCase, but manually constructed patterns can of course be stored and used in the approach.

6 Summary and Outlook

In this paper we have presented our case-based reasoning inspired approach for pattern-based ontology construction, called OntoCase. Initial experiments have already indicated the usefulness of ontology patterns for supporting semi-automatic enterprise ontology construction, but there is still a long way to go before having a completely automatic method for ontology construction. By utilising a full case-based methodology, we suggest as future work to add two more phases in the process, an evaluation and revision phase and a phase of retaining patterns and providing feedback to the pattern base. This research provides a balanced hybrid approach between manual pattern-based ontology construction and reuse, and a completely case-based system. Patterns can be constructed both manually and discovered automatically and they are stored together with information on their confidence.

Key contributions of this research is to improve several of the current drawbacks and issues of existing semi-automatic ontology construction approaches. Further automation reduces the effort to construct an ontology and additionally reduces the need for ontology engineering expertise. At the same time we aim at improving the quality of the output, the constructed ontologies. This is done through introducing both the notion of ontology patterns and additional evaluation and revision steps in the ontology construction process.

The paper describes ongoing research but the first two phases of the approach are already realised, and initial experiments have been conducted showing that automatically constructed ontologies based on patterns do have specific benefits even when compared to hand-crafted ontologies. Although no deep evaluation results can yet be presented for the latest improvements of OntoCase, it adds several steps to improve the output compared to existing OL approaches. It remains to implement the complete OntoCase cycle and conduct further evaluations. We envision that iterative approaches exploiting knowledge reuse and uncertainty, are really the future of semi-automatic ontology construction.

Acknowledgements

This work was partly developed within the MediaILog research project, financed by the Swedish foundation *Carl-Olof och Jenz Hamrins Stiftelse*, and the research project Semantic Structuring of Components for Model-based Software Engineering of Dependable Systems (SEMCO), based on a grant from the Swedish KK-Foundation (grant 2003/0241). Special thanks to the three anonymous reviewers for valuable comments on how to improve this paper.

References

1. Gruber, T.: A translation approach to portable ontology specifications. Knowledge Acquisition 5, 199–220 (1993)
2. Guarino, N.: Formal Ontology and Information Systems. In: Proceedings of FOIS 1998, pp. 3–15 (1998)
3. Uschold, M., King, M., Moralee, S., Zorgios, Y.: The Enterprise Ontology. Knowledge Engineering Review 13, 31–89 (1998)
4. Blomqvist, E., Sandkuhl, K.: Patterns in Ontology Engineering: Classification of Ontology Patterns. In: Proc. of ICEIS2005, Miami Beach, Florida (2005)
5. Blomqvist, E.: Fully automatic construction of enterprise ontologies using design patterns: Initial method and first experiences. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3761, Springer, Heidelberg (2005)
6. Gangemi, A.: Ontology Design Patterns for Semantic Web Content. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 262–276. Springer, Heidelberg (2005)
7. W3C-SWBPD: Semantic Web Best Practices and Deployment Working Group (2004), Available at: `http://www.w3.org/2001/sw/BestPractices/`
8. NeON Website. Available at: `http://www.neon-project.org/`
9. Cimiano, P.: Ontology Learning and Population from Text: Algorithms, Evaluation and Applications. In: Science, Springer, Heidelberg (2006)

10. Fortuna, B., Grobelnik, M., Mladenic, D.: Semi-automatic Data-driven Ontology Construction System. In: Proc. of IS-2006, Ljubljana, Slovenia (2006)
11. Velardi, P., Navigli, R., Cucchiarelli, A., Neri, F.: Evaluation of OntoLearn, a methodology for automatic learning of domain ontologies. In: Ontology Learning from Text: Methods, Evaluation and Applications, IOS Press, Amsterdam (2005)
12. Iria, J., Brewster, C., Ciravegna, F., Wilks, Y.: An Incremental Tri-partite Approach to Ontology Learning. In: Proc. of LREC 2006, Genoa (2006)
13. Kavalec, M., Svatek, V.: A Study on Automated Relation Labelling in Ontology Learning. In: Ontology Learning from Text: Methods, Evaluation and Applications, IOS Press, Amsterdam (2005)
14. Völker, J., Vrandecic, D., Sure, Y., Hotho, A.: Learning disjointness. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, Springer, Heidelberg (2007)
15. Brewster, C., Ciravegna, F., Wilks, Y.: Background and Foreground Knowledge in Dynamic Ontology Construction: Viewing Text as Knowledge Maintenance. In: SIGIR 2003. Proceedings of the Semantic Web Workshop, Toronto, Canada (2003)
16. Fellbaum, C., et al.: WordNet - An Electronic Lexical Database. MIT Press, Cambridge (1998)
17. Shvaiko, P., Euzenat, J.: A Survey of Schema-based Matching Approaches. Journal on Data Semantics IV, 146–171 (2005)
18. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Heidelberg (2007)
19. Cohen, W., Ravikumar, P., Fienberg, S.: A Comparison of String Distance Metrics for Name-Matching Tasks. In: IIWeb 2003. Proc. of IJCAI-03 Workshop on Information Integration on the Web, Acapulco, Mexico (August 9-10, 2003), pp. 9–10 (2003)
20. Chapman, S.: Simmetrics. Available at: http://www.dcs.shef.ac.uk/~sam/simmetrics.html
21. Alani, H., Brewster, C.: Ontology Ranking based on the Analysis of Concept Structures. In: Proceedings of K-CAP 2005, Banff, Alberta, Canada, Canada (2005)
22. Ding, L., Pan, R., Finin, T., Joshi, A., Peng, Y., Kolari, P.: Finding and ranking knowledge on the semantic web. In: Proceedings of the 4th International Semantic Web Conference (2005)
23. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AICom 7, 39–59 (1994)
24. Weber, R.O., Ashley, K.D., Brüninghaus, S.: Textual case-based reasoning. The Knowledge Engineering Review 20(3), 255–260 (2006)
25. Pal, S.K., Shiu, S.: Foundations of Soft Case-based Reasoning. John Wiley & Sons Inc, New Jersey (2004)
26. Frantzi, K., Ananiadou, S., Tsuji, J.: The c-value/nc-value method of automatic recognition for multi-word terms. In: Nikolaou, C., Stephanidis, C. (eds.) ECDL 1998. LNCS, vol. 1513, Springer, Heidelberg (1998)
27. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In: Proc. of ACL 2002 (2002)
28. Blomqvist, E., Öhgren, A., Sandkuhl, K.: Ontology Construction in an Enterprise Context: Comparing and Evaluating two Approaches. In: Proc. of ICEIS 2006, Paphos, Cyprus (2006)
29. Blomqvist, E.: Pattern ranking for semi-automatic ontology construction (submitted, currently under review)

Towards Community-Based Evolution of Knowledge-Intensive Systems*

Pieter De Leenheer and Robert Meersman

Semantics Technology and Applications Research Laboratory (STARLab)
Vrije Universiteit Brussel
Pleinlaan 2, B-1050 BRUSSELS 5, Belgium
{pdeleenh,meersman}@vub.ac.be

Abstract. This article wants to address the need for a research effort and framework that studies and embraces the novel, difficult but crucial issues of adaptation of knowledge resources to their respective user communities, and *vice versa*, as a fundamental property within knowledge-intensive internet systems. Through a deep understanding of *real-time*, community-driven evolution of so-called ontologies, a knowledge-intensive system can be made operationally relevant and sustainable over longer periods of time. To bootstrap our framework, we adopt and extend the DOGMA ontology framework, and its community-grounded ontology engineering methodology DOGMA-MESS, with an ontology that models community concepts such as business rules, norms, policies, and goals as first-class citizens of the ontology evolution process. Doing so ontology evolution can be tailored to the needs of a particular community. Finally, we illustrate with an example from an actual real-world problem setting, viz. interorganisational exchange of HR-related knowledge.

1 Introduction

Collaboration and knowledge sharing have become crucial to enterprise success in the knowledge-intensive European Community and the globalised market world-wide. In this market the trend in innovation of products and services is shifting from mere production excellence to intensive and meaningful knowledge creation and management.

In next-generation computerised distributed working environments, a key objective indeed is to effectively leverage individual competencies of people working together to a *community level*. The World Wide Web has been extremely successful in enabling information sharing among a seemingly unlimited number of people worldwide. It therefore also provides the basic infrastructure that allows on-line virtual communities (professional as well as leisure-oriented) to emerge all around.

Currently, we are witnessing what some call “second-generation Web” (*Web 2.0*), manifested by an explosion of new tools and technologies being developed and shared

* We would like to thank Stijn Christiaens, Aldo de Moor, Tom Mens, Stijn Hoppenbrouwers, and Erik Proper for the valuable discussions about the subject. The research described in this paper was partially sponsored by the EU Leonardo da Vinci CODRIVE project and the EU FP6 IST PROLIX project.

at little or no cost. Social applications like lightweight folksonomies, blogs, wikis, and a plethora of other collaborative tools yield value-added communication platforms that enable virtual communities to emerge that share ideas, knowledge and resources in a usually *self-organising* manner [33]. Even as we limit ourselves (as we do in this article) to professional, "goal-oriented" communities, the logical and inevitable next step is an increase in scale and maturity of such communal knowledge sharing, achieved through collaboration and integration within and between different and diverse communities.

Ontologies, being formal, computer-based specifications of *shared conceptualisations* of the worlds under discussion, are instrumental in this process by providing shared resources of *semantics* [18,17,22]. Such formal semantics are evidently fundamental in the development of any collaborative, knowledge-intensive services, methodologies or systems that claim to capture and evolve, in real time, relevant commonalities and differences in the way communities conceptualise their world and communicate about it. To this end, the *pragmatic aspects* of the exchange of knowledge and information are crucial. Pragmatics represent the intentions, motivations and methodologies of the persons involved and need to become formalised and unambiguous for effective exchange to occur.

This article wants to address the need for a research effort and experimental framework that studies and embraces the novel, difficult but crucial issues of adaptation of knowledge resources to their respective user communities, and *vice versa*, as a fundamental property within knowledge-intensive internet systems. Through a deep understanding of *real-time*, community-driven evolution of so-called ontologies, a knowledge-intensive system can be made operationally relevant and sustainable over longer periods of time.

To bootstrap a framework, we adopt and extend the DOGMA ontology framework, and its community-grounded ontology engineering (OE) methodology DOGMA-MESS, with an ontology that models community concepts such as business rules, actors, roles, norms, and goals as first-class citizens of the ontology evolution process. Doing so ontology evolution can be tailored to the needs of a particular community. Finally, we illustrate with an example from an actual real-world problem setting, viz. interorganisational exchange of HR-related knowledge.

2 Progress Beyond the State of the Art

Several EU FP6 integrated projects[1] and networks of excellence[2] tested and validated a vast number of methods and tools for formalising and applying knowledge representation models in a wide variety of applications [18,17,22]. However, there is still little understanding of, and technological support for, the methodological and *evolutionary* aspects of ontologies as resources. Yet these are crucial in distributed and collaborative settings such as the Semantic Web, where ontologies and their communities of use naturally and *mutually* co-evolve.

[1] E.g., http://www.sekt-project.com, http://dip.semanticweb.org

[2] E.g., http://knowledgeweb.semanticweb.org

2.1 Single User Ontology Evolution

For managing the evolution of domain vocabularies and axioms by one single dedicated user (or a small group under common authority), established techniques from data schema evolution [1,19] have been successfully adopted, and consensus on a generic ontology evolution process model has begun to emerge [21,25]. In Fig. 1, we illustrate a single user, hence *context-independent* change process model, based on [2], that distinguishes four activities, over three phases in the change process model: *initiation*, *execution*, and *evaluation*. For a comprehensive state-of-the-art survey on ontology evolution activities we refer to [7].

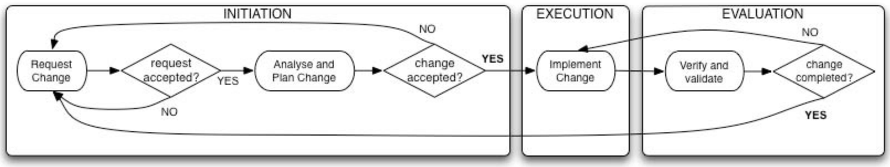


Fig. 1. A context-independent change process model

Initiation. *Requesting* the change has to do with initiating the change process. Some human stakeholder or automatic discovery process [30] wants to make a change to the ontology under consideration for some reason, and will post a so-called change request. Usually a change request is formalised by a finite sequence of elementary change *operations* [1]. The set of applicable change *operators* to conduct these change operations is determined by the applied knowledge representation model.

Planning the change has to do with understanding *why* and *where* the change needs to be made. Therefore, a crucial part of this activity has to do with *change impact analysis*, which is "the process of identifying the potential consequences (side effects) of a change, and estimating what needs to be modified to accomplish a change" [3]. This is very helpful to estimate the required cost and effort (see [27] for a business view on ontology engineering costs). A result of this activity may be to decide to implement the change, to defer the change request to a later time, or to ignore the change request altogether.

Execution. The execution of a change request should have transactional properties, i.e., atomicity, consistency, isolation, and durability [16]. Our process model realises these requirements by strictly separating the change request specification and subsequent implementation, as suggested by [30]. Implementing a change is a difficult process that necessitates many different sub-activities: *change propagation*, *restructuring* and *inconsistency management*. Furthermore, different *evolution strategies* might be implemented to resolve inconsistencies during a change operation [1,31,23].

Evaluation. The last, but certainly not the least, activity in the change process has to do with *verification* and *validation*. Verification addresses the question "did we build the product right?", whereas validation addresses the question "did we build the right product?". A wide scale of different techniques has been proposed to address these questions, including: testing, formal verification, debugging, and quality assurance.

2.2 Collaborative Ontology Engineering

Collaboration aims at the accomplishment of shared objectives and an extensive coordination of activities [26]. In order to create synergy in the result of the collaborative OE process, the *socio-technical* aspects of the community play a very important role [8]. E.g., through implicit and explicit norms, the authority for the control of the process is distributed among many different participants.

In a collaborative setting, there are many additional complexities that should be considered. As investigated in FP6 integrated projects[3] on collaborative networked organisations, the different professional experiences; social and cultural backgrounds among communities and organisations can lead to misconceptions, leading to frustrating and costly ambiguities and misunderstandings if not aligned properly. This is especially the case in interorganisational settings, where there may be many pre-existing organisational sub-ontologies, inflexible data schemas interfacing to legacy data, and ill-defined, rapidly evolving collaborative requirements [10]. Furthermore, participating stakeholders usually have strong individual interests, inherent business rules, and entrenched work practices. These may be tacit, or externalised in workflows that are strongly interdependent, hence further complicate the conceptual alignment.

Summarising, one should not merely focus on the practice of creating ontologies in a project-like context, but view it as a continuous process that is integrated in the operational processes of the community. In a collaborative setting, the *shared background* of communication partners is continuously negotiated as are the characteristics or values of the concepts that are agreed upon. The shared background is externalised as formal *artefacts*, which can be ontological elements of various levels of granularity, ranging from individual concepts of definitions to full ontologies, contributing to the communal knowledge. This includes contributed taxonomies, concept definitions, interfaces, workflow definitions, etc. The evolution of this shared background should be *orchestrated* by and *grounded* in the community.

2.3 Towards Community-Based Evolution of Knowledge-Intensive Systems

Successful virtual communities and communities of stakeholders are usually self-organising. The knowledge creation and sharing process is driven by implicit community *goals* such as mutual concerns and interests [24]. In order to better capture relevant knowledge in a *community-goal-driven* way, these community goals must be externalised appropriately. E.g., in [5], we identified several *macro-level* ontology engineering *processes* that (in a particular methodological combination) provide the goal of the ontology engineering process: *lexical grounding and disambiguation*, *specialisation*, *integration* (including *negotiation*), *axiomatisation*, and *operationalisation*. However, in their operational implementation, which we call OE *micro-processes*, methodologies differ widely. In order to link the community goals to relevant *strategies* underlying the collaborative ontology engineering process and its support, we are required to model relevant community aspects (i.e. establish their formal semantics), and ultimately integrate the concept of community as first-class citizen, where possible, in the evolution processes of the knowledge-intensive system.

[3] E.g. http://ecolead.vtt.fi/

This *holistic* approach is breaking with current practice, where systems are usually reduced to only its IT aspects, with the possible exception of the field of *organisational semiotics* (e.g., MEASUR [29]) and the *language/action perspective* (e.g., RENESYS [8]) that already involved a few socio-technical aspects of communities such as norms and behaviour in legitimate user-driven information system specification [12].

3 Requirements for Our Framework

Based on our observations above, we now make some assumptions in order to proceed to a design for a knowledge-intensive system that supports community-based ontology evolution.

3.1 A Constructivist Approach

Humans play an important role in the interpretation and analysis of meaning during the *elicitation* and *application* of knowledge. Hence, given the diversity and the dynamics of knowledge domains that need to be accommodated, a viable ontology engineering methodology should not be based on a single, monolithic domain ontology that presumes a unique *objective* reality, that is maintained by a single knowledge engineer. It should instead take a *constructivist* approach where it supports multiple domain experts in the gradual and continuous externalisation of their *subjective* realities contingent on relevant formal community aspects [5].

Technically, this requires a knowledge engineering methodology that supports the collaborative building and managing of increasingly mature versions of *contextualised* ontological artefacts (conceptualising their divergent subject realities), and of their *inter-dependencies*. Ultimately, this will allow human experts to focus on the subtle "community-grounded" meaning alignment *negotiation* processes.

3.2 Modelling of Communities: Norms and Negotiation

The RENISYS method [12] conceptualises community information system specification processes as *conversations for specification* by relevant community members. It therefore uses formal *composition norms* to select the relevant community members who are to be involved in a particular conversation for specification. Next, it adopts a formal model of conversations for specification to determine the acceptable conversational moves that the selected members can make, as well as the status of their responsibilities and accomplishments at each point in time.

Similarly for our purposes, by grounding evolution processes in terms of community aspects such as composition norms and conversation modes for specification, the knowledge-intensive system can be precisely tailored to the actual needs of the community [9]. In next paragraphs, we first identify some composition norms, and then show conversation modes will play a role in meaning negotiation.

Composition Norms. Among other community aspects that will orchestrate the collaborative OE processes, in this paper we only distinguish between two kinds of composition norms: (i) *external norms* that *authorise* relevant actors in the community for

an action within a particular ontological context, and (ii) *internal norms* that, independently from the involved actors, *constrain* or *propagate* the evolution steps, enforced by the dependencies the involved ontological context has with other contexts.
Inspired by Stamper [29] and de Moor [12], an external norm is defined as follows:

> **if** precondition **then** actor **is** {permitted/required/obliged}
> **to** {initiate/execute/evaluate} action **in** ontological context.

The *precondition* can be a boolean, based on a green light given by an entitled decision organ, or triggered by some pattern that detects a trend or inconsistency in the actual ontological structures. The *deontic status* states whether an *actor* is permitted, obliged, or required to perform a particular *role* (initiation, execution, validation) within the scope of a certain *action* (e.g., a micro-level or macro-level OE process). A micro-level process is an operation conducted in terms of micro-level primitives, e.g. *introduceConcept*, *defineGenus*, etc. In [5], we defined such a set of OE primitives for characterising context dependencies (see Sect. 4.3).
An internal norm is defined as follows:

> {initiate/execute/evaluate} action **in** ontological context **is constrained to**
> $\{ \bigcup_i primitive_i(e_i^1, \ldots, e_i^n) \}$ **where** $\forall_i \{e_i^j, \ldots, e_i^k\} \in$ *ontological_context*$_i$
> $(1 \leq j \leq k \leq n)$.

Performing a particular action role in some ontological context is (in order to perform that action) constrained to use a restricted toolbox of primitives ($\bigcup_i primitive_i$) of which some parameters are bound to ontological elements $e_i^j, \ldots, e_i^k$, that were already grounded in some ontological contexts. In Sect. 5, we will extensively illustrate the above definitions.

Meaning Negotiation. The constructivist approach engenders meaning divergence in the respective organisational contexts. This requires a complex socio-technical *meaning negotiation* process, where the meaning is aligned. However, sometimes it is not necessary (or even possible) to achieve context-independent ontological knowledge, as most ontologies used in practice assume a certain professional, social, and cultural perspective of some community. The key is to reach the *appropriate* amount of consensus on *relevant* conceptual definitions through *effective* meaning negotiation in an *efficient* manner [11]. As suggested earlier, a negotiation process is defined as a specification conversation about a concept (e.g. a process model) between selected domain experts from the stakeholding organisations. For an excellent survey on different conversation models we refer to [9].

3.3 Design for a Framework

The constructivist approach to ontology engineering in complex and dynamic realistic settings threatens to slip back into out-of-control evolution processes, when the *socio-technical* aspects are not well understood. Furthermore, these rapidly evolving community aspects, and the many dependencies they have with the actual knowledge artefacts in the knowledge structures, lead to knowledge structures that can be extremely volatile.

Hence, research into a special-purpose and comprehensive framework will be needed to address the manageable evolution of knowledge structures, while respecting the autonomous yet self-organising drives inherent in the community.

We now bootstrap a design for a *community-driven knowledge-intensive system* (KIS):

1. The technical part of KIS, including a general ontology server and an API that provides collaborative *elicitation*, *representation*, and *analysis* functionalities for knowledge artefacts and context dependencies.
2. The social part of KIS, representing the client communities where communication and norms form the basis for coordinated goal-oriented action.
3. The community-grounded *meaning evolution support system* (MESS) part orchestrating the co-evolution cycle between between community communication and their knowledge.

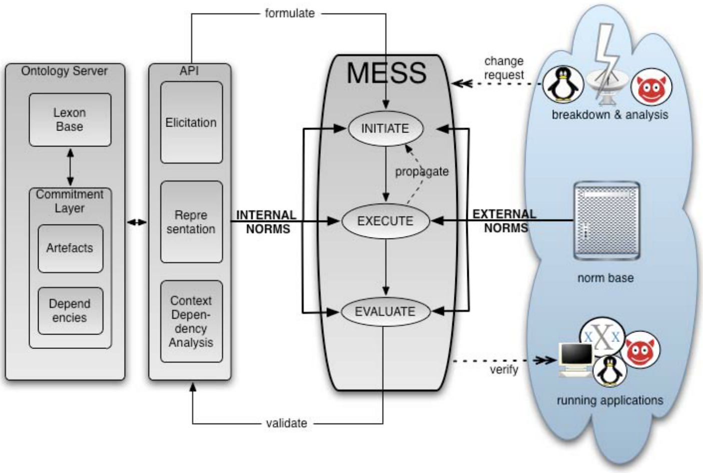


Fig. 2. A design for a knowledge-intensive system

Figure 2 illustrates the three parts and the co-evolution cycle as follows: (i) the evolution process starts with some individual stakeholders becoming aware of a communication mismatch, that causes a work breakdown. Next, (ii) this breakdown is described in a concrete request for eliciting the relevant consensus to reinstate normal community communication. The knowledge administrator analyses and formulates the change request into concrete macro-level OE processes, which in turn can be decomposed into micro-level processes. Each of the micro-level processes engage a MESS process, which process-wise is similar to the change process model described in Sect. 2.1. Additionally, the MESS process is coordinated by the relevant participating members that are selected from the external norm base, and the impact of the changes is calculated from the formal dependencies defined by the internal norms.

For the technical part of KIS, we adopt the DOGMA ontology framework, which we present next.

4 DOGMA Ontology Engineering

Ontology is an approximate shared *semiotic* representation of a subject matter. The DOGMA [22] ontology approach and framework is adopted with the intention to create flexible, reusable bounded semiotics for very diverse computational needs in communities for an unlimited range of *pragmatic* purposes [34].

The DOGMA approach has some key distinguishing characteristics that make it interesting for our purpose, such as (i) its groundings in the linguistic representations of knowledge, (ii) the explicit separation of the *conceptualisation* (i.e., lexical representation of concepts and their inter-relationships, materialised by so-called *lexons*) from its *axiomatisation* (i.e., *semantic* constraints) and (iii) its independence from a particular representation language. The goal of this separation, referred to as the *double articulation* principle [28], is to enhance the potential for re-use and design scalability. Lexons are initially uninterpreted binary fact types, which increases their potential for reusability across community perspectives or goals. The axiomatisation of lexons guarantees the specification needed for semantic consistency and well-formedness in a particular collaborative context (see further). Lexons are collected in the Lexon Base, a reusable pool of possible vocabularies. A lexon is a 5-tuple declaring either (in some *elicitation context* G) [5]: (i) a taxonomical relationship (*genus*): e.g., $\langle G$, $manager$, $is\, a$, $subsumes$, $person\rangle$; or (ii) a non-taxonomical relationship (*differentia*): e.g., $\langle G$, $manager$, $directs$, $directed\, by$, $company\rangle$. Next, we will elaborate more on the notions of elicitation context (Sect. 4.1) and application context (Sect. 4.2).

4.1 Language Versus Conceptual Level

Another distinguishing DOGMA characteristic is the explicit *duality* (orthogonal to double articulation) in interpretation between the *language* level and *conceptual* level. The goal of this separation is primarily to disambiguate the lexical representation of terms in a lexon (on the language level) into concept definitions (on the conceptual level), which are word senses taken from lexical resources such as WordNet [13]. The meaning of the terms in a lexon is dependent on the context of elicitation [5].

E.g., consider a term “capital”. If this term was elicited from a typewriter manual (read: elicitation context), it has a different meaning (read: concept definition) than when elicited from a book on marketing. Hence, we denote:

$$concept(\langle typewritermanual, capital\rangle) \neq concept(\langle marketingbook, capital\rangle).$$

Within a context of elicitation, lexons are not merely syntactic by nature, but underspecified, what makes them reusable for being applied in a specific collaborative *application context* [34] within a UoD. The formal account for application context is manifested through the selection and interpretation of lexons in ontological commitments, and the context dependencies between them.

4.2 Ontological Commitments

The pragmatic account for knowledge artefacts is formalised in *ontological commitments*. Committing to the Lexon Base in the context of an application means selecting a meaningful set S of lexons from the Lexon Base that approximates well the intended vocabulary, followed by the addition of a set of semantic constraints, or rules, to this subset. The result, called an ontological commitment, is a logical theory of which the models are first-order interpretations that correspond to the intended task(s) for achieving a particular goal with a certain level of trust and quality. An important difference with the underlying Lexon Base is that commitments are internally unambiguous and semantically consistent. Ontologies can differ in syntax, semantics, and pragmatics, yet they all are built on these shared vocabularies in the Lexon Base. Examples of ontological commitments include goal, process and communication models, but also business rules, database constraints, or norms.

4.3 Context Dependency Management

Context dependencies constrain the possible relations between the entity and its context, and constrain or propagate the evolution steps within and between different ontological contexts, throughout the ontology engineering processes. Many different types of context dependencies exist, within and between ontological elements of various levels of granularity, ranging from individual concepts of definitions to full ontologies. In, [5], we formalised and illustrated three different types of context dependencies within one ontology (*intra-ontological*) and between different ontologies (*inter-ontological*): articulation (ART), application (APP), and specialisation (SPE). A typical example of a dependency type is the *specialisation dependency*, that exists between a concrete task description and a task template. In order to be complete, the task description should address the specialisation of all, and only those, *differentiae* (plural of differentia) and concepts in the template.

Context dependencies will be used to enforce internal norms. In order to constrain the applicable evolution steps, context dependencies also keep a *change log* in terms of applied micro-level primitives. Next, we will illustrate internal and external norms in real-world example.

5 Community-Based Evolution of an HR Knowledge-Intensive System

In order to illustrate the possibilities of our framework we consider an example that is inspired by the several research projets in the HR domain we are currently involved in[4]. Figure 3 illustrates a snapshot of the scenario, were multiple layers of ontological contexts mutually constrain each other with context dependencies.

– All ontologies commit to an extendible repository (lexon base) of *reusable competence definitions* (RCDs) (lexons). We adopt the definitions as proposed by the HR-XML consortium: an RCD is *a specific, identifiable, definable, and measurable knowledge, skill, ability and/or other deployment-related characteristic (e.g. attitude, behavior, physical ability) which a human resource may possess and which*

[4] http://www.codrive.org (EU Leonardo da Vinci); http://www.prolixproject.org (EU FP7 IST PROLIX); http://cvc.ehb.be/PoCeHRMOM/Frameset.htm (IWT TETRA PoCeHRMOM).

is necessary for, or material to, the performance of an activity within a specific business context[5]. This repository also provides a set of *canonical* relationships between RCDs, including meronymical relationships, i.e. an RCD might be a *facet* or *part of* another RCD.

- The *upper interorganisational* layer includes the $SHARED_{TH}$ ontology, which defines a contributed taxonomy on RCDs.
- On the *lower interorganisational* level, several so-called (governmental) *occupational information networks* accommodate specific collaborative contexts by further "articulating" the taxonomy in $SHARED_{TH}$. For example, O*NET[6] provides a particular classification of skill RCD types. The ART (articulation) dependency between the subcontexts O^*NET_{TH} and $SHARED_{TH}$ enforces the reuse *policy* (which is an internal norm) that all RCD skill types t_{RCD} (e.g., $Basic\,Skill$) introduced in the lower O^*NET_{TH} context must be articulated by some term g_{RCD} (e.g., $Skill$) in the upper $SHARED_{TH}$ context. This policy is denoted as follows:

  execute $introduceRCD(t_{RCD})$ **in** O^*NET_{TH} **is constrained to**
  $\{\ articulateConcept(\langle O^*NET_{TH}, t_{RCD}\rangle, c),$
  $defineGenus(\langle O^*NET_{TH}, t_{RCD}\rangle, \langle SHARED_{TH}, g_{RCD}\rangle)\ \}$
  where $g_{RCD} \in SHARED_{TH}$,

  where c is some concept definition. Other examples of such networks include the Flemish Social-Economical Council[7] (SERV), and the US Army Military Occupational Specialties (MOS) List[8]. They also are expected to follow this community policy when eliciting new RCD types.
- Various higher and lower level *organisational* levels (e.g., O^*NET_{RCM} in Sect. 5.2, $MOS_{Template}$ in Sect. 5.1), including branches within organisations, commit to lower or upper interorganisational levels. In the example below, $Army$ is an organisation consisting of several lower level branches[8] such as humanitarian, armor, aviation, medical service corps, etc. (not illustrated).

RCDs are lexically grounded and disambiguated into concept definitions, however there still are underspecified for particular pragmatic purposes. This specification happens by defining RCD maps.

5.1 Defining RCD Maps

Organisations that commit to occupational information networks such as O^*NET, reuse RCDs and further specify (in various ways) their semantics by combining them in *reusable competency maps*[9] (RCMs), and possibly axiomatise these RCMs. Figure 4 illustrates a reusable competency map for RCD "written expression", that was extracted from the O*NET RCM subcontext "1.A.1.a.4". The APP dependency between O^*NET_{TH} and O^*NET_{RCM} enforces the policy that when building new RCMs for an RCD t_{RCD} (e.g., $Written\,Expression$) in the context of O^*NET, one should not

[5] http://ns.hr-xml.org/2_5/HR-XML-2_5/CPO/Competencies.html
[6] http://online.onetcenter.org/
[7] http://www.serv.be
[8] http://www.us-army-info.com/pages/branches.html
[9] As defined by the HR-XML consortium.

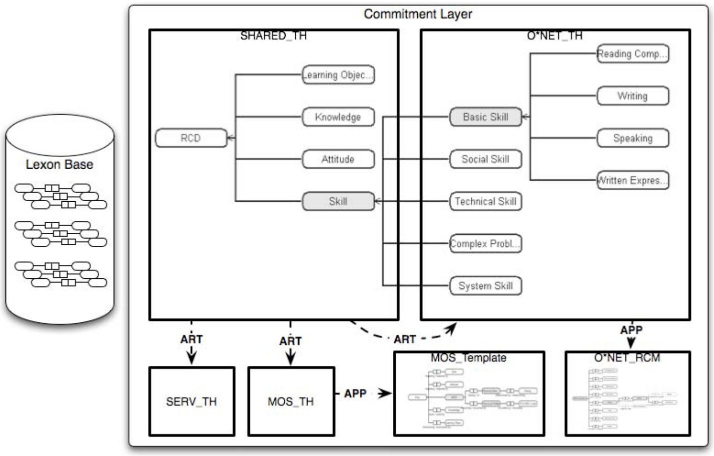


Fig. 3. A snapshot of the scenario in the DOGMA framework: on the left the lexon base, and on the right the commitment layer, were multiple levels of ontological contexts mutually constrain each other with context dependencies

introduce new RCD types, but merely reuse existing RCDs $t_2^{d_i}$ (e.g., $Understanding$) from O^*NET_{TH} in new differentiae ($d_i = \langle O^*NET_{RCM}, t_{RCD}, r_1^{d_i}, r_2^{d_i}, t_2^{d_i}\rangle$). This internal norm is denoted as follows:

execute $buildRCM(t_{RCD}, \bigcup_i d_i)$ **in** O^*NET_{RCM} **is constrained to**
$\{\ defineDiff(O^*NET_{RCM}, d_i, \langle O^*NET_{TH}, t_{RCD}\rangle, \langle O^*NET_{TH}, t_2^{d_i}\rangle\ \}$
where $t_{RCD}, t_2^{d_i} \in O^*NET_{TH}$.

Although all RCMs might be built on the same RCD base repository, they differ widely in structure and semantics, contingent on the subjective perspectives of the different organisational contexts.

5.2 Defining Occupation Specifications

Now consider following scenario where a new military occupational specification (MOS) for "social worker" is to be introduced in the Army. The request for eliciting a new "social worker" MOS is produced in response to a breakdown in achieving a new military strategic goal towards deploying more *humanitarian operations*. These operations come in many forms, requiring HR related to confidence-building measures, power-sharing arrangements, electoral support, strengthening the rule of law, and economic and social development.

The knowledge administrator analyses and formulates the request into concrete OE processes, that are relevant to reach the appropriate amount of consensus about the new MOS in the most effective way. Two important processes are to lexically ground the term in MOS_{TH}, and to analyse the semantics of "social worker". Organisational

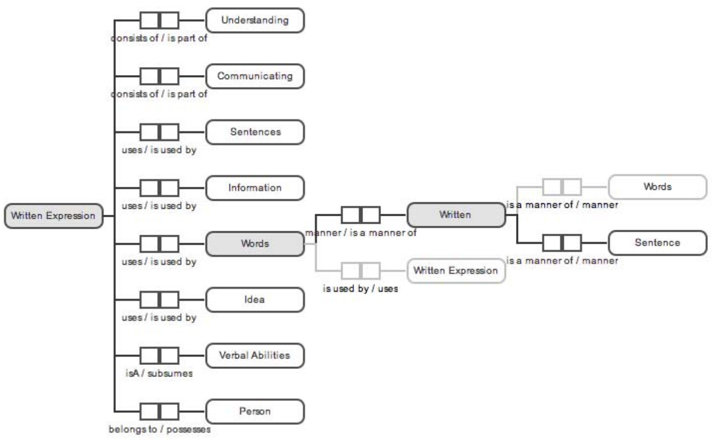


Fig. 4. RCM for "written expression" in the collaborative context O^*NET_{RCM}

policy requires any MOS to be *semantically analysed* according to the MOS template[10] (see Fig 5), which basically consists of two parts:

1. a general description for required skills and attitude, used knowledge, and envisioned learning objectives, specified in terms of artefacts such as upper shared RCMs or organisationally shared specifications;
2. a set of physical requirements, to be assessed with medical evidence data.

The MOS template was elicited by core domain experts and represents the current focus of the community. By specialising a MOS template in terms of reusable interorganisational RCMs it can share its call for HR, and hence attract candidates from other military organisations that have more specialised HR in humanitarian operations such as the United Nations Peacekeepers, or from civilian sectors, as the MOS is not restricted to soldiers, but also include police officers, and other civilian personnel.

Figure 6 illustrates the whole process:

1. The community is aware of a collaboration breakdown, and identifies the need for a new military occupational specialisation for " social worker" as one of the solutions.
2. **initiation:**
   – this breakdown is described in a concrete change request for eliciting the relevant consensus to reinstate normal community communication.
   – if the request is accepted, the authorised knowledge administrator analyses the change request, and formulates it into concrete macro-level OE processes. Fig. 6 only illustrates this for the semantic analysis activity.
   – next, he plans the change. First, he locates the (lower organisational) collaborative context in which the analysis is to be performed, viz. $HumanOps$. Then, based on this information, he calculates the change impact. Therefore, he consults the internal norms. It turns our that in this case there are no dependent

[10] http://www.us-army-info.com/pages/mos/air-defense/14j.html

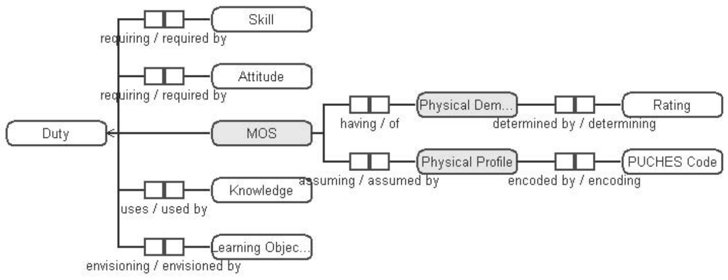


Fig. 5. A template for a military occupational specialisation (MOS) in context $MOS_{Template}$

artefacts. However, as multiple members are authorised to perform each their semantic analysis of social worker, an additional negotiation process to align the resulting divergent specifications will be required.

3. **execution:**
   – once the plan is approved, it moves to the execution phase. Following norm obliges all recruiting officers (ROs) of all branches b to execute their semantic analysis activity for "social worker" in their individual subcontexts $HumanOps_{RO_b}$:

     if initialised(SemanticAnalysis) **then** $\forall_b RO_b$ **is** obliged **to** execute $SemanticAnalysis(\langle MOS_{TH}, social\, worker\rangle, HumanOps_{RO_b})$ **in** $HumanOps_{RO_b}$.

     The internal norms further constrain them to be all specialisations of the MOS template (SPE dependency between $HumanOps_{RO_i}$ and $MOS_{Template}$), and reuse RCD and RCM vocabulary from O^*NET, $SERV$ or MOS (APP dependency between $HumanOps_{RO_i}$ and O^*NET, $SERV$ and MOS). The result is a set of divergent specifications for "social worker". The execution is facilitated by providing the officers with an editing window that is precisely tailored to the job.
4. **evaluation:** The specialisations are evaluated by, e.g. defining a test population for the concepts and relationships, or by committing the organisational data schemas to them.

5.3 Community-Based Meaning Argumentation and Negotiation

Despite the context dependencies enforcing RCD or RCM reuse and template policies, our methodology cannot exclude the possibility that policies are ignored, and hence new competency definitions are rigourously introduced ad hoc. We could further force the reuse policy by defining specific norms that would delegate the exclusive rights for defining new RCDs to the HR-XML consortium. However, such exclusive rights would be unacceptable: we have to accept that the community endorses the constructivist approach, were ontologies should be grounded in the community and in the language of the community itself. Similarly to MOS specialisations for social worker, multiple organisations (such as SERV or MOS) will have divergent RCMs for written expression, conceptualised in terms of RCDs from $SHARED$ of $O * NET$, or in terms of their own familiar organisational competency vocabulary to nuance their intensions.

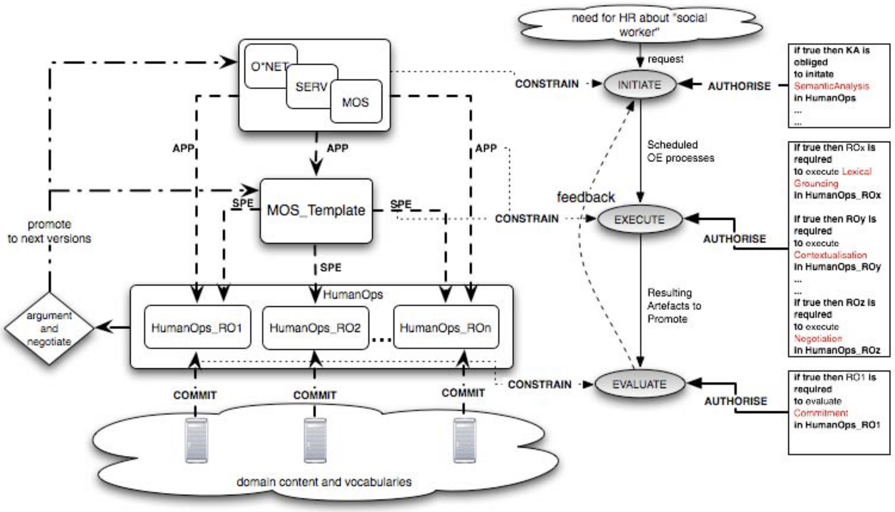

need for HR about "social worker"
request
O*NET
SERV
MOS
APP
APP
APP
MOS_Template
SPE
SPE
SPE
promote to next versions
HumanOps
HumanOps_RO1
HumanOps_RO2
HumanOps_ROn
argument and negotiate
COMMIT
COMMIT
COMMIT
domain content and vocabularies
CONSTRAIN
INITIATE
AUTHORISE
Scheduled OE processes
feedback
CONSTRAIN
EXECUTE
AUTHORISE
Resulting Artefacts to Promote
CONSTRAIN
EVALUATE
AUTHORISE
if true then KA is obliged to initate SemanticAnalysis in HumanOps ... ...
if true then ROx is required to execute Lexical Grounding in HumanOps_ROx
if true then ROy is required to execute Contextualisation in HumanOps_ROy ... ...
if true then ROz is required to execute Negotiation in HumanOps_ROz
if true then RO1 is required to evaluate Commitment in HumanOps_RO1

Fig. 6. Illustration of a collaborative ontology change process

The goal is that organisations can exchange their HR optimally, hence we propose a intermediate solution where organisational ontology engineering processes basically respect the policies enforced by the context dependencies, but are also allowed to introduce competencies from the organisational vocabulary. In any case where the policy is not followed, an alignment process between the stakeholding organisations should bring an acceptable balance between RCD reuse and new organisational competency vocabulary. In [4], we give a semantic account of how RCD reuse can be promoted within the DOGMA approach.

DOGMA-MESS [11] is a constructivist meaning evolution methodology and system, where such a balanced negotiation process is conducted as suggested in the requirements of KIS. In our community-grounded change process, we support DOGMA-MESS in setting up the negotiation agenda automatically: by consulting the internal and external norms, the relevant community members who are to be involved in a particular conversation, and the involved context dependencies can be selected. Ultimately, when consensus is reached, the aligned concept can be promoted and shared to the next version of the upper interorganisational level. It also works the other way around: when some consensus about an artefact is questioned after some validation period, the artefact is mandated to degrade and undergo a new negotiation round. To support the negotiation process several argumentation methods were devised such as HCOME [20] and Diligent [32].

6 Implementation

Currently a first version of a web-based DOGMA-MESS[11] is being tested in several real-world case studies, as illustrated in Sect. 5, and a client variant is being

[11] http://www.dogma-mess.org

implemented in our DOGMA Studio Workbench[12] as we write. Meanwhile, we are installing norm and specification conversation models into the system, and we are planning experiments with other context dependency types (cf. [5]. In [6], we proposed a graph rewriting approach to formalise the semantics of composition norms, and conduct context dependency analysis. This approach promises to be suitable for modelling external norms as well.

7 Discussion and Conclusion

The key challenge of this article was to bootstrap a framework that studies and embraces the novel, difficult but crucial issues of adaptation of knowledge resources to their respective user communities, and *vice versa*, as a fundamental property within knowledge-intensive internet systems. By using norms to select relevant domain experts in OE processes, knowledge evolution is grounded in the community. Furthermore context dependencies, enforces organisations to reuse lower or upper shared ontologies in their local ontological contexts. However, the constructivist MESS process is also democratical in a sense that it allows organisational vocabularies to be introduced, and promoted and shared to the next version of the (upper or lower) interorganisational level. Next we discuss some observations for future research directions.

7.1 Templates

During our experiments, we experienced templates as important instruments in order to conduct knowledge elicitation in a goal-oriented way. E.g., the MOS template reflects the current shared interests regarding the specification military occupations. The template was not predefined, but also co-evolves over time with the actual community interests. In [5], we describe how template evolution triggers a cascade of changes to all its dependent specialisations. In [11] we already give some insights how new trends in the community can be detected by *relevance measures*. Based on the “wisdom of the crowd” principle, if a certain threshold of organisations deviate from the current template, it means there is a trend shift in the knowledge elicitation process in order to serve new interests and goals.

7.2 Internal and External Norms

In this paper, we only modelled a fraction of the community aspects that play an important role in capturing community-grounded knowledge evolution. Amongst other, this will imply other context dependency types, e.g. during the evaluation phase, in order to verify the backwards compatibility of the changed knowledge artefacts with inflexible data schemas interfacing to legacy data.

7.3 Multi-disciplinary Approach

In order to better capture the communication mismatches that cause collaboration breakdown, we have to go wider than current practice by taking explorations of new

[12] http://www.starlab.vub.ac.be/website/dogmastudio

and alternative approaches from multiple relevant disciplines. For example, the field of *communication modelling* and *discourse analysis* [14] has applied communication theories that are the basis for inter-organisational and inter-personal communication acts and knowledge exchange. These concepts can be used for the analysis of communication processes present in any kind of information and knowledge exchange and in particular in negotiations. Furthermore, much can be learned from the field of *information system engineering* (in particular collaborative software engineering), model-driven engineering, and *model-driven architecture* [15] offers a wealth of techniques and tools for versioning, merging and evolving artefacts. Naturally, as already mentioned, principles from the field of *organisational semiotics* can be useful in modelling communities and identifying community aspects in ontology evolution.

7.4 Human-Computer Confluence

Clearly, many of the ontology engineering activities are intrinsically interactive in nature and require a lot of human intervention. This does not mean, however, that we should rule out other approaches that are fully automated. A careful balance and communication is needed between human, semi-automatic (i.e. requiring human interaction) and automatic approaches for knowledge interpretation and analysis processes. Ultimately, communities will consist of a mix of human and software agents that transparently will communicate and request services from each other in order to maintain the shared knowledge structures appropriately.

References

1. Banerjee, J., Kim, W.: Semantics and implementation of schema evolution in object-oriented databases. In: ACM SIGMOD Conf., SIGMOD Record, pp. 311–322. ACM Press, New York (1987)
2. Bennett, K., Rajlich, V.: Software maintenance and evolution: a roadmap. In: ICSE - Future of SE Track, pp. 73–87 (2000)
3. Bohner, S., Arnold, R.S.: Software Change Impact Analysis. IEEE Computer Society Press, Los Alamitos (1996)
4. Christiaens, S., De Bo, J., Verlinden, R.: Competency model in a semantic context: Meaningful competencies (position paper). In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006. LNCS, vol. 4278, pp. 1100–1106. Springer, Heidelberg (2006)
5. De Leenheer, P., de Moor, A., Meersman, R.: Context dependency management in ontology engineering: a formal approach. LNCS Journal on Data Semantics 8, 26–56 (2006)
6. De Leenheer, P., Mens, P.: Context-driven ontology engineering: a graph rewriting approach. In: Proc. of Agtive, Kassel, Germany, Springer, Heidelberg (2007)
7. De Leenheer, P., Mens, T.: Ontology Management for the Semantic Web, Semantic Web Services, and Business Applications, from Semantic Web and Beyond: Computing for Human Experience. In: Ontology Evolution: State of the Art and Future Directions, Springer, Heidelberg (2007)
8. de Moor, A.: Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems. PhD thesis, Tilburg University, The Netherlands, ISBN 90-5668-055-2 (1999)

9. de Moor, A.: Language/action meets organisational semiotics: Situating conversations with norms. Information Systems Frontiers 4(3), 257–272 (2002)
10. de Moor, A.: Ontology-guided meaning negotiation in communities of practice. In: Mambrey, P., Gräther, W. (eds.) C&T 2005. Proc. of the Workshop on the Design for Large-Scale Digital Communities at the 2nd International Conference on Communities and Technologies, Milano, Italy (July 2005)
11. de Moor, A., De Leenheer, P., Meersman, R.: DOGMA-MESS: A meaning evolution support system for interorganizational ontology engineering. In: Schärfe, H., Hitzler, P., Øhrstrøm, P. (eds.) ICCS 2006. LNCS (LNAI), vol. 4068, pp. 189–203. Springer, Heidelberg (2006)
12. de Moor, A., Weigand, H.: Formalizing the evolution of virtual communities. Inf. Syst. 32(2), 223–247 (2007)
13. Fellbaum, C. (ed.): Wordnet, an Electronic Lexical Database. MIT Press, Cambridge (1998)
14. Fortuna, G.M., Mladenic, D.: System for semi-automatic ontology construction. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, Springer, Heidelberg (2006)
15. Djuric, D., Gasevic, D., Devedzic, V.: Model Driven Architecture and Ontology Development. Springer, Heidelberg (2006)
16. Gray, J.: The transaction concept: Virtues and limitations (invited paper). In: Proceedings of Very Large Data Bases, 7th International Conference, pp. 144–154. IEEE Computer Society Press, Los Alamitos (1981)
17. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition 5(2), 199–220 (1993)
18. Guarino, N.: Formal ontology and information systems. In: FOIS 1998. Proc. of the 1st Int'l Conf. on Formal Ontologies in Information Systems, pp. 3–15. IOS Press, Amsterdam (1998)
19. Katz, R.H.: Towards a unified framework for version modeling in engineering databases. ACM Comput. Surv. 22(4), 375–408 (1990)
20. Vouros, G.A., Alonso, J.P., Kotis, K.: Hcome: tool-supported methodology for collaboratively devising living ontologies. In: Bussler, C.J., Tannen, V., Fundulaki, I. (eds.) SWDB 2004. LNCS, vol. 3372, pp. 155–166. Springer, Heidelberg (2005)
21. Maedche, A., Motik, B., Stojanovic, L.: Managing multiple and distributed ontologies on the semantic web. The VLDB Journal 12(4), 286–302 (2003)
22. Meersman, R.: The use of lexicons and other computer-linguistic tools in semantics, design and cooperation of database systems. In: CODAS 1999. Proc.of the Conf. on Cooperative Database Systems, pp. 1–14. Springer, Heidelberg (1999)
23. Mens, T., Van Der Straeten, R., D'hondt, M.: Detecting and resolving model inconsistencies using transformation dependency analysis. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 200–214. Springer, Heidelberg (2006)
24. Nonaka, I., Takeuchi, H.: The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation. Oxford University Press, Oxford (1995)
25. Plessers, P.: An approach to web-base ontology evolution. PhD thesis, Vrije Universiteit Brussel, Brussel (2006)
26. Sanderson, D.: Cooperative and collaborative mediated research. In: Harrison, T.M., Stephen, T. (eds.) Computer networking and scholarly communication in the twenty-first century, pp. 95–114. State University of New York Press (1994)
27. Simperl, E., Sure, Y.: Ontology Management for the Semantic Web, Semantic Web Services, and Business Applications, from Semantic Web and Beyond: Computing for Human Experience. In: The Business View: Ontology Engineering Costs, Springer, Heidelberg (forthcoming)
28. Spyns, P., Meersman, R., Jarrar, M.: Data modelling versus ontology engineering. SIGMOD Record 31(4), 12–17 (2002)
29. Stamper, R.: Linguistic Instruments in Knowledge Engineering. In: Language and Computing in Organised Behaviour, pp. 143–163. Elsevier Science Publishers, Amsterdam (1992)

30. Stojanovic, L.: Methods and Tools for Ontology Evolution. PhD thesis, University of Karlsruhe, Germany (2004)
31. Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-driven ontology evolution management. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 285–300. Springer, Heidelberg (2002)
32. Tempich, C., Pinto, S., Sure, Y., Staab, S.: An argumentation ontology for distributed, loosely-controlled and evolving engineering processes of ontologies. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 241–256. Springer, Heidelberg (2005)
33. Van Damme, C., Hepp, M., Siorpaes, K.: Folksontology: An integrated approach for turning folksonomies into ontologies. In: Proceedings of the ESWC Workshop Bridging the Gap between Semantic Web and Web 2.0, Innsbruck, Austria, Springer, Heidelberg
34. Zhao, G.: Application semiotics engineering process. In: Maurer, F., Ruhe, G. (eds.) SEKE, pp. 354–359 (2004)

ImageNotion: Methodology, Tool Support and Evaluation

Andreas Walter and Gábor Nagypál

FZI Forschungszentrum Informatik, Haid-und-Neu-Straße 10-14, 76131 Karlsruhe, Germany
awalter@fzi.de
disy Informationssysteme GmbH, Erbprinzenstr. 4-12, Eingang B, 76133 Karlsruhe, Germany
nagypal@disy.net

Abstract. The content of image archives changes rapidly. This makes the traditional separation of ontology development and image annotation steps no longer feasible. In this paper, we present an approach, termed ImageNotion that allows for the collaborative development of domain ontologies directly by domain experts with minimal ontology experience. ImageNotion is both a methodology based on the idea of the ontology maturing process model, and the name of the tool supporting this methodology. ImageNotion embeds the creation of ontology entities, termed imagenotions, into the work process of creating semantic annotations of images and their parts. Both the creation of imagenotions and the creation of image annotations are visual, user friendly processes, implemented by a web application that integrates all of the required functionality in one consistent framework. Besides the theoretical concepts, this paper also presents the results of our evaluation of the system with experienced image annotators and librarians having minimal ontology background.

Keywords: Collaborative ontology development, semantic image annotation.

1 Introduction

In state-of-the-art systems for the management and retrieval of image contents ([7], [16]), tagging is used for the annotation of images. Tags are very easy to use for everyone, but have many shortcomings. Because tags are simple, unrelated words, synonyms or the same concepts in different languages are represented by unrelated tags, and the same tag represents all concepts of a word with homonymous meanings. This of course hurts retrieval performance.

The semantic annotation of images allows semantic search. Compared to tag-based systems, some interesting features can be provided by semantic search, such as proposing semantically similar images to a search request or automatically refining the search request considering the current search context. In addition, semantic search delivers better results than tag-based search. This gives enough motivation to provide semantic search engines for the retrieval of images.

Semantic image annotation requires the usage of ontology elements. This leads to some new problems compared to the tag-based annotation of images. While tagging is very easy and collaboratively usable for all kind of users, ontologies appear very complicated to most users [11]. Additionally, the traditional workflow to allow semantic annotation is not very practical. There are two separated work processes: one is the ontology development by ontology experts (knowledge engineers and domain experts), and the other is the semantic annotation of images by the owners of image contents[1]. Every time when users require new ontology elements, they have to apply for the extension of the ontology. Especially when contents in image archives change very fast (e.g. Flickr has more than thousand new images per minute[2]), this separation is not feasible in practice because it is time consuming and cost intensive.

The owners of images have the knowledge about their image contents, and also about the domain of the image. For a high quality of semantic image annotations, they are motivated to participate in the creation of ontologies. Therefore, we propose the collaborative creation of ontologies embedded in the semantic annotation process of images. This empowers the owners of images to do both, to create semantic image annotations and to develop the ontology itself.

The main challenge to reach this goal is to make the ontology development process so easy that even the average user, who has minimal or even absolutely no ontology experience, should be able to meaningfully extend the ontology. Further, the ontology development process should also be collaborative so that users may profit from the work already done by their fellow users. In this paper, we introduce our visual ImageNotion approach that provides high usability and simplicity. ImageNotion embeds the collaborative development of ontologies into the work process for the semantic annotation of images. Our approach is motivated by the success of Web 2.0 applications for image annotation. ImageNotion is implemented by an easy to use, collaborative and web based application.

The paper is organized as follows: Section 2 presents the state-of-the-art of semantic image annotation and discusses its problems for practical usability. Section 3 analyzes the requirements for combined ontology development and semantic image annotation. Section 4 discusses related work. Section 5 presents the ImageNotion formalism; Section 6 presents the ImageNotion methodology. Section 7 introduces the ImageNotion tool. Section 8 presents the results of our evaluation and Section 9 concludes the paper and provides some outlook.

2 State-of-the-Art for Semantic Image Annotation

2.1 Usual Workflow

The usual workflow for semantic image annotation is introduced in [10] and [12] as follows: during semantic image annotation domain ontologies are accessed, and the image is annotated with the proper ontology elements. In a real world setup, this usually means two different work processes (see Fig. 1): ontology experts (knowledge engineers and domain experts) are responsible for the development and maintenance

[1] Domain experts and annotators are sometimes the same individuals.

[2] http://www.flickr.com/photos

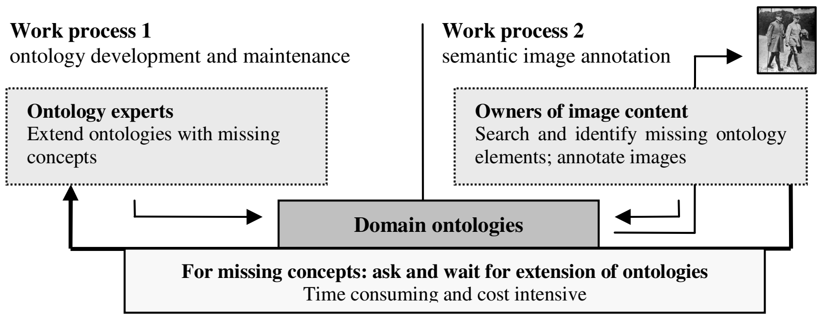

Work process 1
ontology development and maintenance
Work process 2
semantic image annotation
Ontology experts
Extend ontologies with missing concepts
Owners of image content
Search and identify missing ontology elements; annotate images
Domain ontologies
For missing concepts: ask and wait for extension of ontologies
Time consuming and cost intensive

Fig. 1. Traditional work flow for semantic image annotation

of ontologies [5] and the owners of image contents (aka. users) are responsible for the semantic image annotation.

2.2 Problems of the Usual Workflow

The usual workflow for semantic image annotation has the following four main problems in practice:

- **Separation of processes:** The separation of ontology development and semantic image annotation is very inefficient. Users themselves can not extend or change ontology elements when it is required. Instead, they have to ask ontology experts and wait until they extend the ontologies. Besides the time factor this also requires a lot of communication because it is the annotator who has the required context for the extension of the ontology, and this contextual knowledge has to be transferred to the side of the ontology experts. Especially in the case of fast changing contents it is not feasible anymore.
- **Complicated formalisms:** Users must understand the meaning of ontology elements they use. This fails in most cases because of too complicated modeling constructs (ontology formalisms), too complicated, extensive or abstract ontology design and incomplete ontology documentation [6].
- **Complicated tools:** The tools for semantic image annotation are very sophisticated and not very usable for most users, especially for non-ontology-experts.
- **Missing support for collaboration:** Ontology building is not a one time activity but a continuous, evolving process. Ideas and understanding emerge implicitly in daily work and mature only gradually through the interaction with others to explicit formal and shared conceptualizations. The communication needed for the successful maturing of users understanding of the domain is usually not supported adequately.

2.3 Our Approach: Work Integrated Ontology Development

To overcome these problems we propose a collaborative ontology development process that is embedded into the work process of semantic image annotation. Our approach that implements this idea is called ImageNotion. In the following section, we define the requirements for this approach and introduce the the ontology maturing process model that describes the process of collaborative ontology development.

3 Collaborative and Work Integrated Ontology Development

3.1 Advantages of Collaborative Ontology Development

Collaborative ontology development saves a lot of time and costs compared to the traditional separated ontology development by ontology experts. Time is saved because users do not need to talk to the ontology experts in order to extend the domain ontologies and they do not have to transfer their context that motivated the ontology extension. Costs are saved because collaborative ontology development reduces the number of required ontology experts for ontology design and maintenance, and because both ontology experts and users need less time to accomplish their work. Moreover, the development of ontologies in bigger communities is not only faster, but also results in a better common understanding of the created ontologies because of the extensive documentation and the protocol of the discussion about the major design decisions. Just consider the collaborative creation of contents in Wikipedia[3], the same effect may be expected at the collaborative creation of ontologies when the process is adequately supported by tools.

3.2 Requirements for a Collaborative Ontology Development

In [1], we have defined the requirements for a collaborative ontology development in general. Based on these results, we will now give an overview on the important requirements for collaborative ontology development focused on semantic image annotation:

- **Usability and simplicity:** Collaboration assumes that users take part in community activities. This requires the lowering of barriers for the given activity. In the context of ontologies, this means the reduction of their complexity and formality. Therefore, tools and work steps must be informal, lightweight, easy-to-use and easy to understand.
- **Work integration:** Users should be able to change seamlessly between semantic image annotation and ontology development – otherwise, they fail in annotating their images as desired. This requires the integration of both tasks in one framework.

During our evaluation we noticed that users usually start ontology development with concrete, tangible concepts, and they model more abstract concepts only later. Hence, a collaborative ontology development methodology should support the bottom up strategy – in contrast to the top down (or middle-out) strategy that is proposed by most ontology development processes [5].

3.3 The Ontology Maturing Process Model

The ontology maturing process model [1] describes collaborative ontology development as interconnected individual learning processes [13]. As the ontology

[3] http://www.wikipedia.org

matures, individual image annotators learn to identify entities in their image contents and learn how to express them with ontology elements. The processes are interconnected because they operate on the same ontology (and sometimes even on the same images).

The ontology maturing process model describes four different steps in the learning process. Focused on semantic image annotation, these steps are:

- **Step 1: Emergence of new ideas.** Every time when a user identifies missing concepts for semantic annotation of added images, elements denoting these new concepts are introduced to the ontology. In this step, concepts are described informally, e.g., by textual tags.
- **Step 2: Consolidation in communities.** During collaborative usage of the new ontology elements, the ideas are refined, incorrect or unhelpful ideas are rejected.
- **Step 3: Formalization:** For image annotation, this means the creation of relations between ontology elements, or creating rules for the creation of new descriptive information.
- **Step 4: Axiomization:** In this step, background knowledge for improving inferencing processes, e.g. for query answering, is added. For most users, this step is too complicated and confusing. Therefore, this step is normally done by knowledge engineers, mostly with specialized tools for axiomization, e.g. Protégé [15].

It is very important to stress that the ontology as a whole will usually contain parts that have different maturing grades. I.e., an ontology may contain parts that are already formalized or even axiomatized, but may also have new parts that are just emerging. The ontology maturing process model allows the semantic annotation of images with ontology elements from all maturing grades.

We believe that the ontology maturing process model describes the real world process of ontology development closely and therefore we require that an ontology development methodology (including ours) should be compatible with this model.

4 Related Work

Our work contributes in the following areas: methods and tools for collaborative ontology development, and tools for semantic image annotation. An important feature is that we address the combination of these two aspects. Currently, we are not aware of any other work that achieves this goal. Therefore we can only review works that addresses partial fields of our research.

Semantic annotation of images: *Flickr* [7] and *Riya* [16] allow browser-based annotation of images with tags in a work integrated environment. However, semantic annotation of images using ontologies is not possible. In these collaborative applications, everyone may add tags. Additionally, *Riya* allows the annotation of image parts and supports automated face detection. *Photostuff* [10] is a stand-alone application that allows the semantic annotation of images with imported ontologies. Ontology development and collaborative work is not possible with this tool.

Tool support for collaborative ontology development: *Protégé* [15] is one of the most popular tools for ontology development. With an extension (*Collaborative Protégé* [21]), it also allows collaborative work. *KAON* OIModeler [22] and *OntoEdit* [20] provide collaborative features, as well. The problem of these tools is that they are too complicated for non-ontology-experts. *Semantic Wikis* [23] are browser-based and easy to use. However, they share the common drawback with the mentioned ontology editors that a separate tool is required for semantic image annotation and therefore ontology development is no more work integrated.

Methodologies and tools for automated ontology development: Automated ontology development can help in cases, when images are already annotated with tags or free-text. In such cases, these methods can help build adequate ontologies. The *OntoGen* [8] tool uses text mining to cluster tags and a question and answer system to semi-automatically generate ontologies. *FolksOntologies* [4,16] uses statistical methods to generate ontologies and enhances tags for example with synonyms from *WordNet* [24]. In most cases, automatically generated ontologies require manual corrections and extensions. For new images without tags, we think that it is better to use approaches that allow the semantic annotations directly. This reduces the required work compared to an approach that first requires tagging and later the manual correction of automatically generated ontologies.

Work integrated ontology maturing: *SOBOLEO* [25] allows the work integrated maturing of ontologies even for non-ontology-experts. However, it is intended for the annotation of web pages and not for the annotation of images and especially image parts. M*yOntology* [19] describes a similar approach using wikis.

5 The ImageNotion Formalism

Our approach is the result of our search for something more intuitive and more understandable for users to support the ontology development process and to allow the semantic annotation for all users. The result is a visually supported formalism and an ontology development methodology called ImageNotion.

5.1 Definition of an Imagenotion

The basis of our ontology formalism is called *imagenotion.* An imagenotion (formed from the words image and notion) graphically represents a semantic notion through an image. Our motivation was the ancient observation that "*a picture worth a thousand words*". Furthermore, similarly to many existing ontology formalisms, it is possible to associate descriptive information with an imagenotion. A part of the descriptive information is textual information: labels in different languages (such as English or German). For each language, one of these synonymous labels is selected as the main label of an imagenotion. Other labels are termed synonyms. Additionally, date information (exact date or time interval) allows for the search for images based on an exact date or time interval. Also, it is possible to add links to web pages for an imagenotion. Links help in maturing an imagenotion: with the background information from web pages, users get new information that supports the maturing of existing imagenotions.

5.2 Modeling Ontologies Using Imagenotions

The distinctions between concepts and instances are hard to understand for most users. In our opinion, the distinction between concepts and instances is not intuitive. The difficulty of deciding whether something should be modeled as an instance or a concept is described in [18] and [14]. Here, the notion “ape” may be viewed both as an instance of the concept “species”, or as a concept that has instances such as “Amy, the gorilla”. This difficulty motivates metamodelling features in some ontology formalisms such as OWL-Full [9].

Using imagenotions, users do not need to understand this somewhat artificial separation of ontology elements. Our solution makes no distinction between concepts and instances. However, to give users a better understanding of how to use relations, we introduced two different types of imagenotions, called concrete and abstract ones. Concrete imagenotions represent tangible entities of the real world, such as a specific event, object, or person like “Romano Prodi” (president of EU Commission 1999-2004). Abstract imagenotions represent intangible notions such as “apes” or “president”. This distinction between abstract and concrete imagenotions allows the introduction of the following rule: Use concrete imagenotions for the semantic annotation of images if possible. Then, abstract imagenotions can be automatically inferred for an image, if proper relations among imagenotions were specified in the ontology.

5.3 The ImageNotion Formalism

Because imagenotions are associated with images, they are very intuitive and meaningful internationally, as an image has the same meaning in different languages[4]. Our aim was to keep our formalism as simple and understandable as possible. The ImageNotion formalism currently offers three types of relations that were motivated by the SKOS specification [2]:

- **Broader:** transitive relation to create “is-a” relations. e.g. the relation imagenotion “ape” is connected with this relation with the imagenotion “species”.
- **Narrower:** transitive relation, the inverse of the broader relation.
- **Unnamed:** a generic, symmetric relation between two imagenotions. E.g. a relation from “Romano Prodi” to the “EU Commission”.

5.4 Reasoning and Querying Imagenotions

Our formalism does not allow negative statements, such as “an ape is not a fish”. This makes reasoning in the ImageNotion formalism very easy, because no model checking for inconsistent statements is required. Whenever a user creates relations between imagenotions, or annotates an image with an imagenotion, the system checks

[4] This claim does not hold for imagenotions on a high abstraction level, such as “freedom”.

whether the relation or the annotation can be inferred based on the actual content of the ontology. If yes, the new relation or annotation is rejected. During ontology browsing or semantic search for images, the system can retrieve all imagenotions that are (possibly indirectly) connected with a given imagenotion through broader, narrower or unnamed relations. It is also possible to retrieve all images that are (possibly indirectly) annotated with a given imagenotion. We will evaluate whether further functionalities are required by the content owners to create high quality image annotations, or by image searchers to get high quality search results. E.g., we expect that reasoning with time intervals may be useful.

6 The ImageNotion Methodology

The aim of the ImageNotion methodology is to guide the process of creating an ontology that consists of imagenotions. The main steps of this methodology (see Fig. 2) are based on the ontology maturing process model.

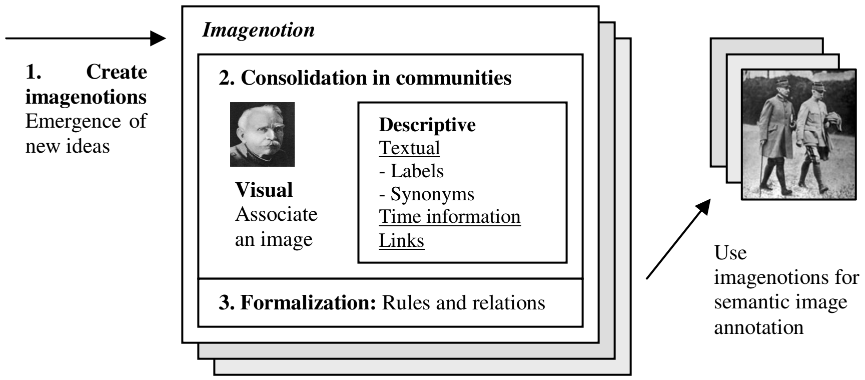


Fig. 2. The ImageNotion methodology

6.1 Ontology Maturing with Imagenotions

Based on the ontology maturing process model, imagenotions iteratively mature in three different steps. In step 1, new imagenotions are created by users in isolation.. These imagenotions are consolidated in Step 2. This phase covers the collaborative reuse of good ideas and the rejection of bad ideas. The members of the community refine the imagenotions by adding (or removing) synonyms, date information and links to web pages such as Wikipedia articles. Step 3 is the formalization of imagenotions. This step is described in the next section in more detail.

6.2 Maturing Processes for Descriptive Information and Relations

The maturing of an imagenotion follows a learning process in two different areas (see Fig. 3). The first step is the maturing of descriptive information; the second step is the maturing of relations. Iterative steps on both levels make an imagenotion mature until a stable, collaborative accepted state emerges.

Maturing process of descriptive information

The maturing process of descriptive information follows the first three phases of the ontology maturing process:

1. **Emergence of ideas:** Creation of the imagenotion by defining a label and an image.
2. **Consolidation:** Collaborative usage and editing of descriptive information.
3. **Formalization:** Definition of rules for the usage of descriptive information, e.g. that each person should include the date information of his or her birthday.

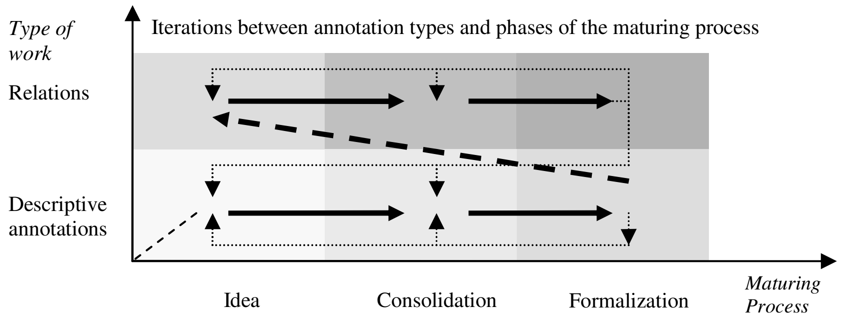


Fig. 3. Maturing process of an imagenotion

Maturing process of relations

After iterative steps for editing descriptive information, the community switches to the next type of work: the creation and editing of relations. Again, this work type follows the first three phases of the ontology maturing process:

1. **Emergence of ideas:** Creation of relations to other imagenotions
2. **Consolidation:** Collaborative acceptance of available relations.
3. **Formalization:** Specification of the relation type, e.g. changing from "unnamed" to "broader".

6.3 Overview of the Complete Workflow for Semantic Image Annotation

Fig. 4 gives an overview of the complete workflow of the combined ontology development and semantic image annotation process using imagenotions. First, the user searches for available imagenotions. If a required imagenotion is missing, or its description is incomplete, the user first participates in the ontology maturing process. Then, she uses the mature imagenotion for the semantic annotation of images. These work steps collaborative iterate. Of course, users may also remove incorrect semantic annotations from images or may directly use already mature imagenotions (according to the user's opinion) for the semantic annotation of images without editing the ontology.

6.4 Semantic Annotation of Images Using Imagenotions

Imagenotions are used for the semantic image annotation instead of textual tags as in other systems. It is easy to see the advantage: all the shortcomings of tagging approaches, e.g. using the same tag for homonyms and different, independent tags for synonyms are solved using ImageNotion because we can provide semantic search instead of full-text search.

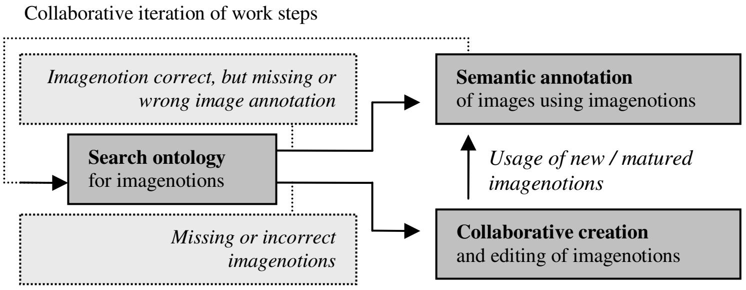

Collaborative iteration of work steps
Imagenotion correct, but missing or wrong image annotation
Search ontology for imagenotions
Missing or incorrect imagenotions
Semantic annotation of images using imagenotions
Usage of new / matured imagenotions
Collaborative creation and editing of imagenotions

Fig. 4. Combined workflow for semantic image annotation and ontology maturing

According to the ImageNotion methodology, it is possible to add imagenotions from every phase of the maturing model and use them for the semantic image annotation. In our experiments, we have made interesting observations about how users improve the semantic annotation of images. These observations support our theory that the semantic annotation of images is an intermingled process of ontology development and image annotation.

From concrete to abstract imagenotions: During semantic image annotation, users first identify concrete things in images and therefore use or create concrete imagenotions, e.g., for concrete persons, events, or objects. Then, they identify abstract imagenotions (e.g., the profession of a person such as “president”) and add them to the images.

Usage of relations to reduce annotation time: In most cases, users annotate some images with similar content and then they switch to ontology development. The motivation for that is to decrease the required work for annotating similar images in the future by creating relations between concrete and abstract imagenotions (see Fig. 5). This improves their workflow as follows: for other, similar images, they only need to add the concrete imagenotion to the image and the information about the relevant abstract imagenotions is inferred automatically using the given relation.

As an example let us assume that users annotate images showing “Romano Prodi”. First, users create the concrete imagenotions for “Romano Prodi” and “EU commission” and use them to annotate their images. Then, they create the abstract imagenotion “president” and use it to annotation images, too. Later, to save work for similar images, they create relations from the imagenotion “Romano Prodi” to the

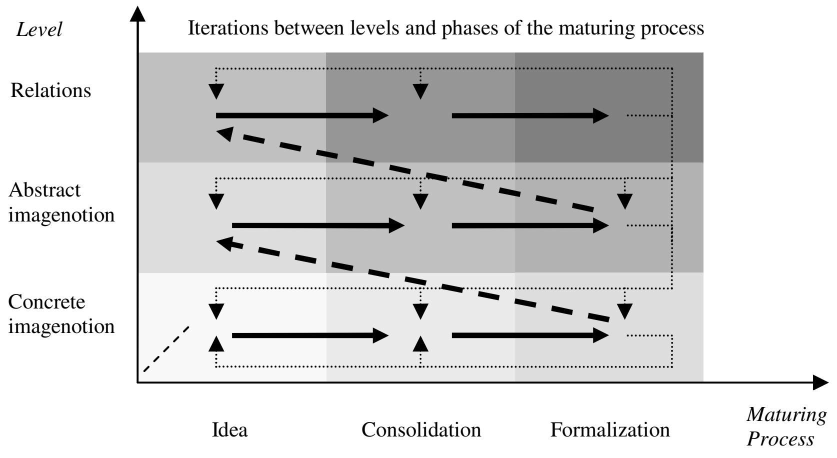


Fig. 5. Maturing process of the semantic annotation of images

imagenotions "president" and "EU commission". Then they use only the imagenotion "Romano Prodi" as semantic annotation for similar images instead of using three imagenotions each time.

7 The ImageNotion Tool

The imagenotion tool supports the combined ontology maturing and the semantic annotation of images using the ImageNotion methodology. We present the features of the expert-based version of the ImageNotion tool.

7.1 User Groups

In the area of professional image archives there are two different user groups: content owners and image searchers. Content owners are responsible for the annotation of their own images. This paper described the expert-based version of the ImageNotion tool, intended for professional image archives. It allows the collaborative work for image editors (content owners) in image archives. They can collaboratively annotate images. Each of them can propose new imagenotions, manipulate or delete them. As the number of experts is normally relatively small, they can have face-to-face discussions, outside the imagenotion system. Thus, in this version collaboration means that all experts view and edit the same global ontology.

Image searches cannot edit the ontology or change image annotations. They can only search for images and browse the ontology.

7.2 Implementation

Our main objective for the ImageNotion tool was to achieve high usability and simplicity. As most users are familiar with the "drag & drop" metaphor, we

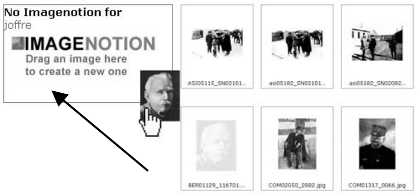
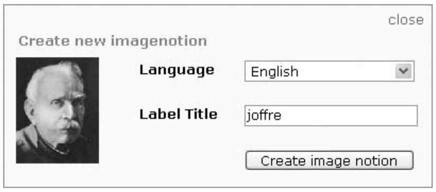


Fig. 6. Create new imagenotion



Language: English

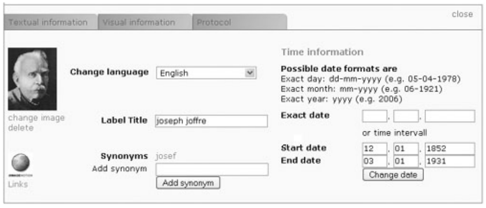
Label Title: joffre

Fig. 7. Insert label text in imagenotion

consistently use this metaphor in our application. In a browser-based application, this requires the usage of Java Script in combination with AJAX. As Wicket[5] allows the flexible integration of both, we selected it as our Java-based web framework.

7.3 Creation of New Imagenotions

Images that contain new concepts require the creation of new imagenotions. In Fig. 6, a user has new images of "Joseph Joffre" (a French general in World War I). Let us assume that the ontology did not contain an imagenotion for Joffre so far, and therefore a new imagenotion is required. In this case, the user chooses one image in the archive showing Joffre and drags this image to the area which allows creating new imagenotions. Now, she can enter a label in her preferred annotation language and the new imagenotion is created (Fig. 7).



Change language: English

Label Title: joseph joffre

Synonyms: josef

Add synonym:

Time information

Possible date formats are
Exact day: dd-mm-yyyy (e.g. 05-04-1978)
Exact month: mm-yyyy (e.g. 06-1921)
Exact year: yyyy (e.g. 2006)

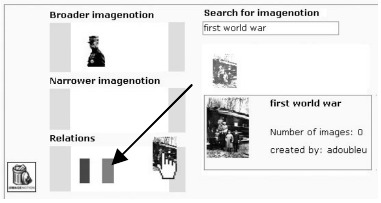
Exact date: . .

or time intervall

Start date: 12 . 01 . 1852

End date: 03 . 01 . 1931

Fig. 8. Edit descriptive information



Search for imagenotion: first world war

first world war
Number of images: 0
created by: adoubleu

Fig. 9. Edit relations

7.4 Editing an Imagenotion and Creating Relations Between Imagenotions

The tool allows the editing of existing imagenotions, too. It is possible to edit the descriptive information (see Fig. 8) and to add unnamed, broader and narrower relations to other imagenotions. To define a relation, the tool first allows searching for other imagenotions. The results may be added as relations, e.g. for "Joseph Joffre", unnamed relations to "France" or to "First World War" may be added (see Fig. 9).

7.5 Semantic Image Annotation

Semantic image annotation is possible with every available imagenotion. First, the user searches for imagenotions and then adds any of the retrieved imagenotions to the list of image or image part annotations by simply dragging it to the list (see Fig. 10).

[5] http://wicket.sourceforge.net

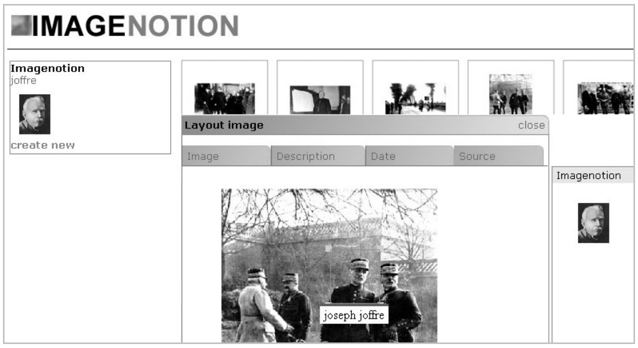


Fig. 10. Semantic annotation of images using imagenotion

7.6 Import of Existing Ontologies

It is not always necessary to build ontologies from scratch. Therefore, our tool allows the import of ontologies that are defined in SKOS [2] or OWL [9]. During the import, each of the defined concepts and instances are converted to an imagenotion. All relations are converted to unnamed relations except those one defined as broader or narrower ones.

8 Evaluation

Our aim was to evaluate whether users who are not ontology experts are able to collaboratively create ontologies and semantically annotate images using the expert-based ImageNotion tool. Therefore, we asked experienced image annotators and librarians who had minimal ontology background to participate in an evaluation workshop. We will now present the setup and results of this workshop.

8.1 Evaluation Setup

Six people (see Table 1) participated in our evaluation workshop which was held in June 2007. One participant (user 1) had well-founded background knowledge about semantic formalisms; two of the participants (users 2 and 3) had much experience with tag-based annotation systems but did not have any experience with semantic

Table 1. Overview on the participants of the evaluation

User	**1**	**2**	**3**	**4**	**5**	**6**
Experience with tag-based annotation	Flickr	Flick Ourweb	Flickr Ourweb	No	No	No
Experiences with semantic formalism	OWL	No	No	Dewey DDC	Dewey DDC	Dewey DDC
Experiences with semantic application	Protégé	No	No	SBN Thesaurus	SBN Thesaurus	SBN Thesaurus
Profession	Text Mining	Assistant in picture library	Translator in picture library	Librarian	Librarian	Librarian
Experiences with computer	Very high	Normal	Normal	Normal	Normal	Normal

formalisms and applications. The other three participants were familiar with thesauri (Dewey DDC [3]), but not with ontologies or with image annotation systems.

For the evaluation, we provided 854 images from two domains: "First World War" and "European politicians". The task for the participants of the evaluation was the semantic annotation of these images. The ontology development thereby had to start from scratch. The workshop took four hours. During the first hour, we presented the ImageNotion tool and its features. In the second hour, the participants used the tool for training purposes and they could ask questions about its features. After that, we removed all semantic image annotations and imagenotions created so far. The following results are based on all imagenotions and semantic image annotations that were created during the remaining two hours of the workshop.

8.2 Overall Result

Table 2 shows the overall work steps for the two different work processes: ontology maturing and semantic image annotation. In the following, a work step means one user action, e.g. annotating an image, creating or maturing an imagenotion. Altogether, the participants have created 46 imagenotions and edited them in 115 work steps. Thus, the number of work steps for the ontology maturing is higher than the 110 work steps for the annotation of images (see Table 2). Clearly more concrete imagenotions (35) were created than abstract imagenotions (11). Altogether 46 imagenotions were created.

Table 2. Number of work steps for ontology maturing and semantic image annotation

Work process I: Ontology maturing	
Number of created imagenotions	46
Imagenotions with only one work step	10
Number of work steps	115
Average number of work steps on imagenotions per user	2,5
Average number of work steps per user for the maturing of imagenotions	19,2

Work process II: semantic image annotation	
Number of annotated images	68
Number of work steps	110
Number of used imagenotions for semantic image annotation:	26
Average of imagenotions used for semantic annotation per image	1,6
Average number of image annotations per user	18,3

Additionally, users created 40 relations to other imagenotions. From the 46 imagenotions, 10 imagenotions had only one work step and can be considered obsolete. Some were created as duplicates from other, collaboratively used imagenotion, e.g. "presi" instead of "president". Altogether, 26 imagenotions were used for the semantic annotation of images and image parts. Other nine imagenotions (e.g. "president" or "general") were used indirectly through relations. This supports our theory from section 6.4 that the semantic image annotation benefits from the maturing process from concrete to abstract imagenotions and the usage of relations between them. In future work, we will provide mechanisms that will help to identify obsolete imagenotions so that they can be deleted.

Table 3. Collaborative usage of imagenotions

Number of users	**Number of imagenotion**	**Percent of imagenotions**
1	32	70
2	11	24
3	1	2
4	1	2
5	1	2

Table 4. Annotation types and maturing of imagenotions

Type of annotation	**Annotation**	**Number of imagenotion**	**Percent**
Descriptive	Image defined	33	71
	Label changed	7	15
	Date	12	56
	Link	18	39
	Synonyms	5	11
	Total	75	
Relations		40	87

Already in such a short time of two hours, the participants have collaboratively used imagenotions (Table 3). 24 percent of them were used by two participants; one imagenotion – labeled "First World War" – was even used by five users. Table 4 supports our theory from section 6.2 for the maturing of imagenotions in two different types of work: 75 work steps were used for the work type I for extending the descriptive information of imagenotions and 40 work steps for the work type 2 – the creation of relations to other imagenotions.

Table 5. Aggregated user actions in the several work steps

Relation to abstract imagenotions		2	5	6	2	1	2	2		2	**3**	
Relation to concrete imagenotions		3	2	2	1	3			1		1	1
Descriptive annotation	**46**	**32**	**12**	9	6	8	5	3	2		1	
Semantic image annotation		3	7	**7**	**19**	**11**	**12**	**15**	**11**	**9**	2	**2**
Work step	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**

8.3 Detailed View on User's Interactions

Table 5 shows the aggregated number of different user actions in each work step. We separated the work steps for semantic image annotation and the work steps for the maturing of an imagenotions, which is separated in work steps for editing descriptive information and editing relations. The bold entries show the mostly used actions in each work step. After creating descriptive annotations (steps 1-3), users switch to the semantic annotation of images (steps 4-10) and then back to the imagenotions to add relations (work step 11).

8.4 Survey of the ImageNotion Tool

After the workshop, we asked the participants to fill out a survey of the ImageNotion tool. The average rating for the usability concerning the creation of imagenotions and the semantic annotation of images have a very pleasant average rating (1.8). The problems with the creation of relations (average rating 2.5) seemed to be a problem with the participants understanding of the theory about relations that led to the most questions during and after the workshop. These values altogether are very encouraging for us, but also show the need to improve the ImageNotion tool especially in the area of relations.

Table 6. Survey results for the ImageNotion tool

User	1	2	3	4	5	6	AVG
Creation of imagenotions	1	2	2	2	2	2	1.8
Image annotation	1	2	1	3	2	2	1.8
Creation of relations	1	3	3	3	3	2	2.5

(1) very easy (2) Problems at the beginning (3) Difficult (4) very difficult

8.5 Discussion

Our evaluation showed that our approach of integrating collaborative ontology development into the process of collaborative semantic image annotation is promising: even non-ontology-experts can create semantic annotations and relations using ImageNotion. An interesting observation is that more than half of work is spent for the creation and maturing of imagenotions that are used for the semantic image annotation in the other work steps. This shows that users are qualified to create ontologies if the ontology development process is integrated in their workflow, i.e., if they have the chance to participate in the ontology development.

9 Conclusion and Future Work

We have shown that the usual, separated work processes, i.e., ontology development by ontology experts and the semantic image annotation by content owners, are not feasible in practice – especially not when the content of the image repository frequently changes. As a solution, we proposed ImageNotion – a work integrated, collaborative ontology development methodology and a tool based on the ontology maturing process model. ImageNotion allows each user to participate in the development of ontologies of imagenotions during the semantic annotation of images. The ImageNotion ontology formalism is simpler than classical ontology formalisms and thus lowers the barrier of participation for non-ontology-experts. Thereby, we identified two separated maturing processes – one for the maturing of imagenotions and one for the maturing of semantic image annotations. This observation supports our thesis: a successful semantic annotation of images is only possible with a work integrated approach for ontology development.

Our evaluation showed that the ImageNotion methodology supported by the *expert-based* ImageNotion tool is promising and well accepted by users with minimal ontology experience.

Our future work focuses on larger scale evaluations of the expert-based image notion tool and also on the extension of the tool to support the ontology maturing process for big communities (such as the community of Flickr). Latter requires additional methods and tools, such as the creation of user groups for specific topic of interests, voting, discussion mechanisms about ontology elements and image annotations, and personalization features such as the possibility to define own images for the representation of imagenotions.

Acknowledgement

This work was co-funded by the European Commission within the project IMAGINATION. Also, we want to thank the participants of our evaluation workshop.

References

1. Braun, S., Nagypál, G., Schmidt, A., Walter, A., Zacharias, V.: Ontology Maturing: A Collaborative Web 2.0 Approach to Ontology Engineering. In: WWW 2007. Proc. of the Workshop on Social and Collaborative Construction of Structured Knowledge, Banff, Alberta, Canada (2007)
2. Brickley, D., Miles, A.: SKOS Core Vocabulary Specification. W3C (2005), http://www.w3.org/TR/2005/WD-swbp-skos-core-spec-20051102
3. Dewey, D.D.C.: The Dewey Decimal Classification, http://www.oclc.org/ca/en/dewey/
4. Van Dammer, C., Hepp, M., Siorpaes, K.: FolksOntology: An Integrated Approach for Turning Folksonomies into Ontologies. In: International Workshop at the 4th European Semantic Web Conference on Innsbruck, Austria (June 2007)
5. Fernández-López, M., Gómez-Pérez, A.: A survey on methodologies for developing, maintaining, integrating, evaluating and reengineering ontologies. Deliverable 1.4, EU IST Project IST-2000-29243 OntoWeb (2002)
6. Gómez-Pérez, A.: Handbook of Applied Expert Systems. In: Gómez-Pérez, A. (ed.) Knowledge Sharing and Reuse, CRC Press, Boca Raton (1997)
7. Flickr: Welcome to Flickr – Photo Sharing. http://www.flickr.com/ (accessed: 27.06.7007) (2007)
8. Fortuna, B., Grobelnik, M., Mladenic, D.: Semi-automatic Data-driven Ontology Construction System. In: IS 2006. Proc. of the 9th International multi-conference Information Society, Ljubljana, Slovenia, Slovenia (2006)
9. McGuinness, D., van Harmelen, F.: OWL Web Ontology Language, http://www.w3.org/TR/owl-features/ (accessed: 27.06.7007)
10. Halaschek-Wiener, C., Golbeck, J., Schain, A., Grove, M., Parsia, B., James, A.: Annotation and provenance tracking in semantic web photo libraries. In: International provenance and annotation workshop (2006)
11. Hepp, M.: Possible Ontologies: How Reality Constrains the Development of Relevant Ontologies. IEEE Internet Computing 11(7), 96–102 (2007)
12. Hollink, L., et al.: Semantic annotation of image collections. In: KCAP 2003. Workshop on Knowledge Markup and Semantic Annotation (2003)
13. Maier, R., Schmidt, A.: Characterizing Knowledge Maturing: A Conceptual Process Model for Integrating E-Learning and Knowledge Management. In: Proc. of the 4th Conference Professional Knowledge Management – Experiences and Visions, Potsdam, Germany (2007)
14. Motik, B., Maedche, A., Volz, R.: A Conceptual Modeling Approach for building semantics-driven enterprise applications. In: Meersman, R., Tari, Z., et al. (eds.) ODBASE 2002. LNCS, vol. 2519, Springer, Heidelberg (2002)
15. Protégé. The Protégé Ontology Editor and Knowledge Acquisition System. http://protege.stanford.edu/ (accessed: 27.06.7007)
16. Riya. Riya - Visual search (accessed: 27.06.7007) (2007), http://www.riya.com/
17. Schmitz, P.: Inducing Ontology from Flickr Tags. In: Proc. of the Collaborative Web Tagging Workshop at the 15th WWW Conference, Edinburgh, Scotland (2006)

18. Schreiber, G.: http://www.cs.man.ac.uk/ horrocks/OntoWeb/SIG/challenge-problems.pdf
19. Siorpaes, K., Hepp, M.: myOntology: The Marriage of Ontology Engineering and Collective Intelligence. In: International Workshop at the 4th European Semantic Web Conference on Innsbruck, Austria (June 2007)
20. Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., Wenke, D.: OntoEdit: Collaborative Ontology Engineering for the Semantic Web. In: International Semantic Web Conference 2002, Sardinia, Italia (2002)
21. Tudorache, T., Noy, N.: Collaborative Protégé. In: WWW 2007. Proc. of the Workshop on Social and Collaborative Construction of Structured Knowledge, Alberta, Canada (2007)
22. Volz, R., Oberle, D., Staab, S., Motik, B.: KAON SERVER - A Semantic Web Management System. In: WWW 2003. Alternate Track Proc. of the Twelfth International World Wide Web Conference, Budapest, Hungary (20-24 May, 2003) (2003)
23. Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., Studer, R.: Semantic Wikipedia. In: WWW 2006. 15th international conference, Edinburgh, Scotland (2006)
24. WordNet. A lexical database for the English language, http://wordnet.princeton.edu/ (accessed: 27.06.7007)
25. Zacharias, V., Braun, S.: SOBOLEO – Social Bookmarking & Lightweight Ontology Engineering. In: Workshop on Social and Collaborative Construction of Structured Knowledge, 16th International World Wide Web Conference, Canada (2007)

Optimal Learning of Ontology Mappings from Human Interactions

Sumit Sen, Dolphy Fernandes, and N.L. Sarda

Deptt of Computer Science,
IIT Bombay, Mumbai 400 076, India
{sumitsen, dolphy, nls}@cse.iitb.ac.in

Abstract. Lexical similarity based ontology mappings are useful to obtain semantic translations of database schemas across application domains. Incremental improvement of such mappings can be obtained from human inputs of ontology mapping. Manual mappings are labor intensive and need to be assisted by machine-generated mappings in a semi-automated approach. Heuristics based approaches allow multiple strategies to learn human expertise in concept mappings. Such learning improves the level of automation of the mapping process. We analyze heuristics based Bayesian learning of manual mappings to improve effectiveness of machine-generated mappings. Our results show that human based mappings contribute higher improvement in the machine-generated values of lexical similarity in comparison to those of structural similarity. The optimal weightage for structural similarity learning is inversely proportional to the complexity of given ontology graphs.

1 Introduction

Semantic heterogeneity of databases has received attention for many years now and ontology based mapping has been increasingly viewed as an engineering solution to the problems. Based on specifications of the conceptualizations as a more generic layer above the schema specifications, ontologies serve as an intermediate step to specify and resolve semantics of the contents of a database system. Two categories of semantics can be differentiated in regard to (a) classes or schema names and (b) Individuals or instances of the classes.

While the later is by no means a trivial problem, we state our approach based on semantics of the class or schema names. The requirement of machine learning in this case is similar to that of learning schema matching by humans[1]. However, it is important to understand that certain areas of learning from human experts are more critical than others. The scalability of manual mappings procedure is also a difficult issue and it is important to note the complexity for humans to comprehend semantic implications of the relative position of a class in a complex class hierarchy. This paper aims to explore the optimality of learning from human knowledge in order to achieve higher efficiency of ontology mappings in databases.

2 Background

The Framework for Interoperability of Geospatial Information using Ontologies (FIGO) is developed on the basis of lexical similarities of class names and their attributes. This framework uses specifications of database ontologies based on a graph of layers, classes and their attributes [2]. Layers are aggregates of certain classes grouped as per their usages. Both Layers and Classes can have child elements in the form of Sub-layers and Sub-classes respectively. Attributes can be both XML data-types and also object-types (contains Classes). This graph structure, without further axioms about the classes and attributes themselves, forms the basis of our simplistic ontologies for a given database. The graph representing an ontology of *n* nodes, average depth of hierarchy equal to *p* and average height *q* can be represented as *G*(n,p,q). Given two graphs O_S and O_T, we need to determine a mapping between their nodes. The ontology matching problem now transforms into generating a similarity matrix given by

$$M[O_S X\, O_T] = \{m_{S1T1}, m_{S1T2}, \ldots\ m_{S1Tn} \\ m_{S2T1}, m_{S2T2}, \ldots m_{S2Tn} \\ \ldots\ldots \\ m_{Sm,T1}, m_{SmT2}, \ldots \quad m_{SmTn}\} \tag{1}$$

$$\text{such that } 0 \le m_{ST} \le 1$$

The matrix is directed from a given source to a target and the reversibility of the values is not assumed (i.e. the similarity is not symmetrical). The question that such a mapping answers is ‘which are the classes and attributes of the source ontology that can provide corresponding values for the classes and attributes of the target’. This is similar to the prevalent understanding of ontology mapping in literature. Ehrig and Staab [6] define ontology mapping: “Given two ontologies O_1 and O_2, mapping one ontology onto another means that for each entity (concept C, relation R, or instance I) in ontology O_1, we try to find a corresponding entity, which has the same intended meaning, in ontology O_2.” The mapping assumes that inability of assessing the similarity results in a zero value, and hence such cases are treated as *non-equivalent.* The other relations used in the mappings are *equivalent*, *sub-class*, *super-class* and *attribute union.* Mappings with values lower than a given threshold (*t*) is assumed as *non-equivalent.* In this paper we assume *t* = 0.5.

2.1 Previous Work

The important aspects of the assessment of machine generated values of similarity involve

1. Classes and attributes are represented by their names and short descriptions. Both these are used to assess the lexical similarity from WordNet[3]. The synonym, hyponym and hypernym relations are used to characterize the relations between different classes.

2. Lexical similarity values are propagated based on a directional propagation of similarity. The similarity of attributes, subclasses and super-classes are used to influence the original values of lexical similarity between classes themselves. This approach [2] employs no-penalty based heuristics to ensure directional use of similarity flooding [4].
3. Similarity propagation from subclasses and superclasses are combined to obtain final scores of similarity between a given *source class* and a *target class*. Attribute mappings remain unchanged by the similarity propagation.

The differences between machine generated mappings in comparison to human generated mappings has been compared by Sen *et al*[2]. and it has been reported that there exists opportunities to improve the precision and recall of the machine based mappings. The generated values can be improved with two different approaches, (i) optimization of the heuristics used and (ii) learning of manual mappings. Semi-automatic ontology mapping approach in FIGO involves machine-based mappings generated by different approaches, viz. (i) lexical similarity alone, (ii) subclass and attribute similarity propagation (iii) combined similarity propagation of subclass, super-class and attributes.

2.2 Learning from Manual Mappings

Machine learning has been deployed for schema matching problems. Berlin and Motro [4] have discussed Bayesian approach to learn schema matching from domain experts using an attribute dictionary. Nottelmann and Straccia [5] have discussed the use of a probability-based framework for automatic learning of schema mapping rules. Similarly Melnik *et al* [4] have also pointed that machine learning forms a part of content based technique for schema matching.The main issues in learning of schema matching problems include

(i) Non-availability of mapping knowledge from domain experts.
(ii) Complexity of large scale (industrial) databases and hence their underlying schemas. Learning in such cases is therefore difficult to design.
(iii) Avoiding cases of over fitting.

Approaches to mappings in ontologies have also used Bayesian learning. A literature survey of recent ontology mapping research shows that Mapping Discovery is an area of ongoing work dominated primarily by machine learning and heuristic techniques. In semantic web research, Bayesian learning has been applied to documents on the web [6].

2.3 Research Problem

Given that the core problem of learning ontology mappings from human based inputs we identify the prevalent research problems in this area.

1. Supervised learning of ontology mappings is not always useful because there are many cases where manual mappings are not available or difficult to obtain

2. It follows that manual mappings are less reliable in complex hierarchies where the structural similarities between two concepts in different ontologies are not obvious to the human user.
3. The training dataset for supervised learning is very important and it is necessary to regulate the information that is learnt in each case.

In the context of FIGO, the information gained from manual mappings include (i) Information about valid mappings & (ii) Information about invalid mappings. It is necessary to incorporate both such information and investigate the conditions in which learning is appropriate. This leads to the following research questions (i) What are the optimal conditions in which learning of human based mappings are beneficial to machine based mapping values? (ii) Is learning of structural similarity from manual mappings beneficial and what is the effect of deep ontological structures on the learning process?

Weights are used to obtain answers for the above questions. These weights form the basis of heuristics, which are employed to ensure optimal learning of human expertise for ontology mappings. While the workflow management based on machine generated mappings enable semi-automation of the mapping process, heuristics based learning of manual mappings improve the machine-generated mappings and hence improves the overall automation. The learning process uses human validations of machine mappings as evidence to strengthen the belief in a given machine mapping. Let $\mathrm{P(A)}$ be the *prior probability* of a given mapping (before observing any manual mappings). Since $\mathrm{P(A)}$ represents prior probability that a given mapping is true, we choose to denote the compliment of this value as $\mathrm{P(A^c)} = 1 - \mathrm{P(A)}$. Bayesian learning uses evidences $P(V \mid A)$ to learn *posterior probabilities* as seen in equation 2. Equation 2 represents the Bayes' theorem [31] for any finite partition $\{A_j, j \in J\}$ of Ω and $V > 0$. Here V represents the condition where human validations for the given mapping exist.

$$P(A \mid V) = \frac{P(V \mid A_i) \cdot P(A_i)}{\sum_{j \in J} P(V \mid A_j) \cdot P(A_j)} \tag{2}$$

Proposition 1

For any mapping value with positive feedback = $P(V \mid A)$ and negative feedback = $P(V \mid A^c)$ the posterior probability is given by

$$P(A \mid V) = \frac{P(V \mid A) \cdot P(A)}{P(V \mid A) \cdot P(A) + P(V \mid A^c) \cdot P(A^c)} \tag{3}$$

The notion of optimal learning emerges from the fact that learning of manual mappings is optimal only under certain conditions. These include the type of learning (whether lexical or structural similarity) or complexity of the ontologies (different values of *n* and *m*). To achieve this we apply weights to the evidences from which learning takes place. We define three different weights in the context of the three different properties (1) Overall optimal weight for learning (*w)* (2) Fractional weights for optimal learning of lexical similarity (w_L) and structural similarity (w_S). (3) Fractional weight for learning under a given complexity of the graphs ($w_{[m,n]}$). We assume that the weight of optimal learning for structural similarity $w_S = w_{[m,n]}$. Therefore overall optimal learning weight criteria are obtained by a combination of these weightages represented by *w*.

The optimality condition is achieved when the precision and recall (and hence the F-score) of the machine-based mappings are highest with respect to the ideal case. Ideal case can be the manual mappings of experts[1]. We use the weights discussed above to obtain a weighted sum of the machine generated values and the learnt values (equation 4).

$$\text{Learnt value} = w \cdot P(A \mid V) + (1 - w) \cdot (P(A)) \tag{4}$$

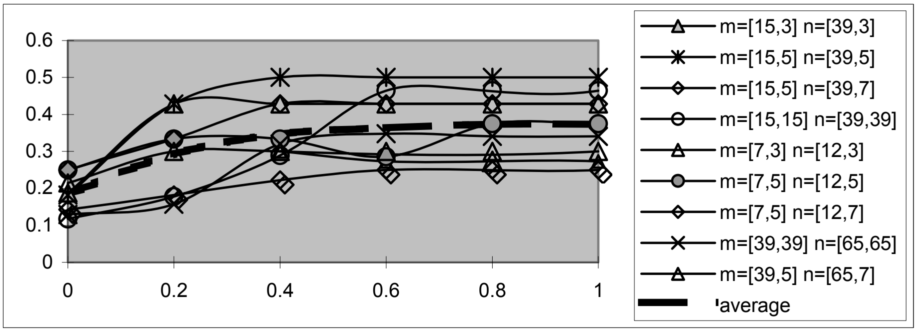
3 Learning Experiments

Our experiments to learn the effects of learning on the performance of machine-generated values of concept mapping were done with two objectives. (1) To assess performance of different weights for learning of mappings based on lexical similarity and structural similarity. Since structural similarity is difficult to measure we use the difference of overall similarity values and lexical similarity. (2) To assess the performance of different weights for learning of mappings for graphs with different number of nodes and average depth. We use the ideal mappings to compute the efficiency of both machine generated and manually generated values. The F-score (harmonic mean of precision and recall) is used as an indicator of the efficiency of the matches. A threshold of 0.5 is used to decide the false positives and false negatives for our experiments.

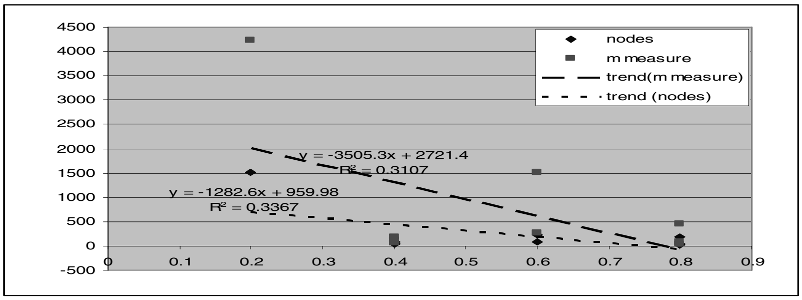
3.1 Machine Learning Employed in FIGO

The plot of the F-measure (harmonic mean of precision and recall) over different weightage given to machine learning is shown in graph 4. This graph shows several cases with different number of nodes and *m* measure. As evident from graph 4, higher complexities of the graph (i.e., bigger [*m,n*] values for both number of nodes and the *m* measure) have a higher correlation with high growth of efficiency in the range of weights [0,0.6]. In all cases, the F-measure of the mappings reach its maximum at $w = 0$ and then remains level. As evident from graph 4, higher complexities of the graph (i.e., bigger *mxn* values for both number of nodes and the *m* measure) have a higher correlation with high growth of efficiency in the range of weights [0,0.6]. In all cases the F-measures of the mappings reach their maximum at a certain *w* and then remain level. We term this weight as w_c.

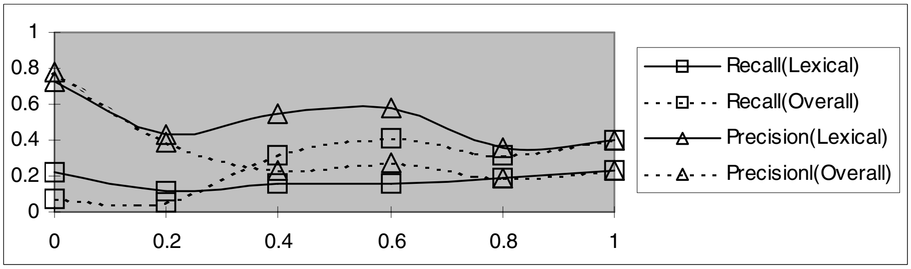
[1] Note that we do not assume that all manual mappings are done by experts. Thus another weight (w_{user}) can also be proposed to have different influence factors for different levels of users.



Graph 1. Changing F-Score values (Y-axis) with respect to different weightages(X-axis used and complexity of the ontology graph (legend). The average value is shown as dashed line and it shows no relative change after *w*=0.



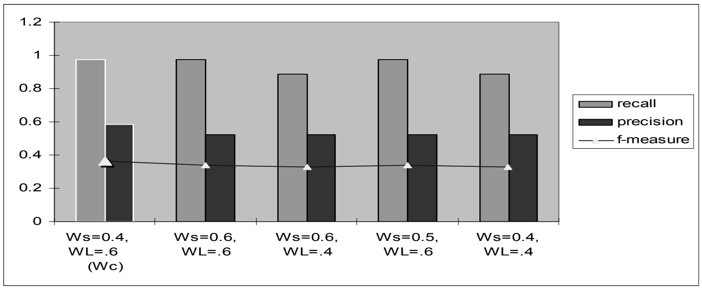
Graph 2. Correlation of optimal weights (w_c) with complexity of ontology graphs (number of nodes and m measure). It shows a negative correlation for both.



Graph 3. Differences in precision and recall (Y-axis) and weightage used for learning human mapping values(X-axis) in relatively large ontology graphs (n>[30,30] and m>[45,45]

The correlation between the structural complexities of a graph with w_c can be seen in graph 2. It shows the variation of w_c with respect to the graph complexity. Graph complexity is shown as a function of (i) number of nodes and (ii) the measure *m* related to average height and weight. It illustrates that with increasing complexity of the ontology graph the required influence of manual mappings goes down. We have discussed that the ontology mapping values obtained from lexical similarity values alone, are different from the overall similarity value, which are based on subclass and super class in the ontology graph. We term this similarity as structural similarity [3]. Structural similarity is introduced by using heuristics based on similarity propagation. The differences in learning manual mapping values based on their type of similarity (either lexical or structural) are also investigated. There are differences between machine learning applied to both lexical similarity and overall similarity. These results are obtained from ontologies of large ontology graphs where human subjects found difficulty in mapping concepts because of the complexity of the graph in which they were present. The precision graph (graph 3) of lexical similarity is seen to be more stable and hence improvement in machine-generated values can largely benefit from a higher weightage factor (w_L) as compared to that for structural similarity (w_S). The correlation of the difference between overall similarity and lexical similarity was seen to be high in most cases where manual mappings failed the ideal cases. The positive correlation of this difference in the two similarity values (and hence the measure of structural similarity) in relation to the number of failed mappings is observed to be around 0.5.

Weights used for learning lexical similarity were found to reach maximum F values at 0.6 where as the w_c value for structural similarity was found to be at 0.4. Based on equation 4 the overall weight for learning is 0.5. The optimal weightage(W_C) for structural similarity (and therefore the overall similarity) is dependent on the size of the graph, as shown in and graph 2. We applied these weights for learning of manual mappings and the results are shown in graph 4. The overall f value obtained for the optimality condition (Ws=0.4,W_L=.6)is higher than those obtained other weightage values . Only cases with comparatively high values of precision and recall have been shown.



Graph 4. Precision, recall and F-measure for different weightages for structural and lexical similarity. The first case is on the basis of the heuristic for optimal learning.

4 Analysis and Discussion

Melnik *et al* [7] has discussed the assessment of automatic matching algorithms and discusses the accuracy of a match result as a measure of labor savings[2] as follows

[2] The measure is not applicable for values of precision less than 0.5 and hence we omit such cases.

$$\text{Accuracy / labor savings} = \text{Recall}\left(2 - \frac{1}{\text{Precsion}}\right) \tag{5}$$

This measure is explained to be pessimistic in nature. The values of accuracy obtained from the results of the heuristics based learning is shown in the table below.

Table 1. Savings from heuristics based learning

Case	Weights	Labor Sav-ings
Optimal	W_S=0.4, W_L=.6 (W_c)	0.288889
Other cases	W_S=0.6 W_L=.6	0.08125
	W_S=0.6, W_L=.4	0.073864
	W_S=0.5, W_L=.6	0.08125
	W_S=0.4, W_L=.4	0.073864

The relative savings in the optimal case is seen to be fairly high, mainly due to the contribution of a higher precision. Learning of manual mappings increases the chances of deleting the false positives because humans are less likely to map dissimilar classes. However the chance of missing a correct match is fairly high in manual mappings. Optimality conditions for learning, in the form correct weightage applies to the learning of lexical similarity and a lower weightage applied to structural similarity (in large ontology graphs), ensures a fairly high precision of ontology mappings. The improvement of machine-generated mappings also provides opportunities for more efficient manual mappings thereafter.

5 Conclusions and Future Work

We have discussed the framework of ontology-based mappings, which learns from manual mappings to improve the machine generated mapping values.

We have explored the optimality conditions of such mappings based on the assumptions of distinctions between lexical similarity and structural similarities between ontologies. The distinction enabled us to experiment with different weightage values for Bayesian learning knowledge about manual mappings. We found that optimal learning ensures higher level of automation (and hence it follows that higher level of automation warrants optimal learning). Our work presents a promising way to ensure high accuracy based ontology mapping techniques in the absence of instance level information. As opposed to techniques that use instances of classes to learn about ontology mappings such as GLUE [7], our technique is applicable in cases where data instances are not available to start with.

We believe that this work is only a beginning in the context of optimal learning of ontology mappings. We believe that the basic principles of heuristics is based learning of human knowledge is applicable in a broader domain of ontology mapping. In the context of the work reported in this paper and our project, some of the directions for future work include (1) Exploring the heuristics employed for learning in further examples to inspect their behavior beyond the geospatial domain currently used. (2) The possibility to increase or modify the lexicon based on the manual mappings is another alternative to increase the accuracy of the machine-based mappings. (3) Human subject testing to explore the critical sizes of ontology graphs in the context of errors produced manually.

Acknowledgements

The work presented in this paper is supported by the NRDMS, Dept. of Science and Technology, Government of India.

References

1. Berlin, J., Motro, A.: Database Schema Matching Using Machine Learning with Feature Selection. In: Bressan, S., Chaudhri, A.B., Lee, M.L., Yu, J.X., Lacroix, Z. (eds.) CAiSE 2002 and VLDB 2002. LNCS, vol. 2590, Springer, Heidelberg (2003)
2. Sen, S., Somavarapu, S., Sarda, N.L.: Class structures and Lexical similarities of class names for ontology matching. In: ODBIS. VLDB Workshop on Ontologies-based techniques for DataBases and Information Systems Co-located with VLDB 2006, Seoul, South Korea (2006)
3. Fellbaum, C.: WordNet. An Electronic Lexical Database. MIT Press, Cambridge (1998)
4. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In: ICDE 2002, IEEE Computer Society, Los Alamitos (2002)
5. Nottelmann, H., Straccia, U.: Information retrieval and machine learning for probabilistic schema matching. In: CIKM 2005, ACM, New York (2005)
6. Ding, Z., et al.: A Bayesian Methodology Towards Automatic Ontology Mapping. In: C&O 2005. AAAI-05 Workshop on Contexts and Ontologies: Theory, Practice and Applications, Pittsburgh, PA (2005)
7. Doan, A., et al.: Learning to Map between Ontologies on the Semantic Web. In: WWW. World Wide Web Confernce (2002)

Automatic Feeding of an Innovation Knowledge Base Using a Semantic Representation of Field Knowledge

Issam Al Haj Hasan[1], Michel Schneider[1], and Grigore Gogu[2]

[1] Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS)
Complexe des Cézeaux, 63173 Aubière CEDEX
{issam.ahh,michel.schneider}@isima.fr
[2] Laboratoire de Mécanique et Ingenieries (LaMI)
Campus de Clermont-Ferrand/les Cézeaux, BP 265, 63173 Aubière CEDEX
grigore.gogu@ifma.fr

Abstract. In this paper, by considering a particular application field, the innovation, we propose an automatic system to feed an innovation knowledge base (IKB) starting from texts located on the Web.

To facilitate the extraction of concepts from texts we distinguished in our work two knowledge types: primitive knowledge and definite knowledge. Each one is separately represented. Primitive knowledge is directly extracted from natural language texts and temporally organized in a specific base called TKB (Temporary Knowledge Base). The entry of the base IKB is the knowledge filtered from the TKB by some specified rules. After each filtering step, the TKB is emptied for starting new extractions from other texts sources.

The filtering rules are specified using variables representing interesting concepts. Their specifications result from the semantics of the innovation operators involved in the innovation process. The variables are initiated from a semantic representation of the operators. The content of the base IKB can be displayed as text annotations. Hence the feeding system is coupled with a user interface allowing the exploration of these annotations through their dynamic insertion in the associated texts.

In this paper, we present the application field and our approach for representing and for feeding the IKB innovation base. We also provide a number of experiment results and we indicate work we plan to undertake in order to improve our system.

Keywords: knowledge base, feeding system, information extraction, semantic representation, conceptual schema, filtering rule, innovation field, innovation operator, semantic network.

1 Introduction

In many fields, there appears the need to build knowledge bases starting from various more or less structured data sources: data bases, documents, web pages.

Very often, the location and the acquisition of these data and knowledge sources are carried out manually. Because of their increasing number, this manual process has to be partially or completely automated. Automatic information extraction from natural language texts (NL texts) was the subject of a great deal of work carried out over the last decade and significant results were obtained [13] and [4].

However, these systems are often specific to one domain and adaptation to a different domain is difficult. Information extraction is generally based on extraction rules involving syntactic and semantic relations between field entities. The manual specification of these rules needs vast experience. The automatic specification can be made by a learning system where the user has to prepare and to tag training texts. Often, the localization of a sufficient number of training examples is time consuming, and in some cases it is impossible.

In the innovation domain, it was very difficult to adapt an existing information extraction system because of the high number and the semantic variety of the innovation operators.

In this paper, we present an approach that we developed to feed an innovation knowledge base (IKB) (cf. section 5.4) by automatically inserting texts examples illustrating the innovation operators (cf. section 3). These examples are retrieved from the NL texts located on the Web and semantically represented in the innovation base. By this representation, users can semantically query the knowledge base and retrieve examples having entities corresponding to concept defining their innovation problem.

In our approach, the extraction rules called “filtering rules” in our work (cf. section 5.5) are specified starting from a semantic representation of the innovation operators. These specifications involve innovation operators entities such as resources, characteristics, actions, and objects presented in section 3. The task of these rules is to semantically filter the relevant information from the texts examples illustrating the operators by locating their specified entities in the natural language texts.

The implemented system (cf. section 5.1) is based on a semantic representation of each innovation operator. In this representation the operator entities are defined by an expert. The expert specifies for each operator entity one or more concepts mapped to a lexical network (for instance WordNet).

In the first stage, the system uses the mapping concepts (cf. section 5.2) and their corresponding terminologies to directly feed the base TKB (cf. section 5.3) from one or more natural language text sources. In the second stage, it uses the semantic representation of each operator to initiate the variables of the filtering rules which are used to feed in the base IKB the relevant knowledge starting from the base TKB.

The system has an interface (cf. section 5.6) allowing users to explore the content of the IKB base by a dynamic insertion of concepts annotations in the corresponding text sources.

2 Information Extraction and Word Sense Disambiguation

Our approach to feed an innovation knowledge base is inspired from two major techniques taken from natural language processing and artificial intelligence: information extraction and word sense disambiguation.

Information extraction (IE) is the task of identifying, collecting and normalizing relevant information from NL texts. It is different from information retrieval (IR) based on a user request to extract the relevant documents from a large collection. But the two techniques are complementary [4].

IE systems don't attempt exhaustive deep analyses of all the text. Rather, they only show interest in the text passages containing relevant information [9]. They are generally based on extraction rules or patterns implying syntactic and semantic relations. Therefore the text must be syntactically analyzed and semantically tagged before applying the rules.

The specification of such rules for free or semi structured texts is a very difficult, time consuming task, and it requires a lot of expertise. For these reasons, the majority of the implemented systems have focused on machine learning approaches [12] such as CRYSTAL [18], WISH [17], and RAPIER [20]. These systems aim to automate the production of the extraction rules and thus to facilitate the adaptation of the system to a new application field.

Word sense disambiguation (WSD), or sense tagging, is the process of assigning the appropriate sense from a lexicon to each word token [21] based on the context. It is an intermediate process and has been used to accomplish most natural language processing tasks. A majority of information retrieval works use this process to compare documents, to index them automatically or to classify and extract documents relative to a user request [6]. This technique can be interesting in IE systems in order to locate the semantic entities and to extract and classify the important text passages.

The WSD works can be broadly classified into supervised and unsupervised techniques. The two techniques are based on external resources such as dictionary, thesaurus or ontologies to achieve their task.

In the dictionary based technique proposed by Lesk [7], semantic closeness is measured by counting the common words appearing in the word context and in the dictionary definitions.

Many works and research programs use WordNet [11] to disambiguate the word sense and specifically to improve automatic text retrieval effectiveness. Most of them propose lexical semantic relatedness measures [2]. For every sense of an ambiguous word, a measure function is applied to select the most appropriate sense.

In our approach, the filtering rules ensure the extraction task. They select the candidate words and disambiguate their senses. This task is carried out using a semantic representation of the interesting knowledge of the application field. This representation allows us to discard the usage of semantic measures. In this work we are not interested in associating each text word to one sense (synset)

[22], but to one field concept. So, the senses to which a word is associated are the senses related to a mapped concept.

3 The Innovation Field and Its Operators

Innovation (or creativity) is a global notion which exists in all scientific, technical, and some business fields [3]. This notion is increasingly being considered as part of new product development. Such development is based on the solving of inventive problems. An inventive problem means a problem that does not have a known solution and can contain contradictory requirements, and solving one problem causes another to appear [10]. Specific methods such as TRIZ [1], [19], WOIS [8], USIT [16], [5] have been elaborated to facilitate education and to contribute to the solving of inventive problems.

Most computer assistance tools for innovation are developed around TRIZ method initiated by G. Altshuller. In this method, he classifies innovation patents by their problem solving process. Consequently, the solving processes are classified into three semantic groups of operators: inventive principle, scientific effect, generic innovating solution. This grouping is based on semantic entities such as resource, characteristic, function where:

– Resource can be a substance (liquid, solid, gas...), a force (attraction, repulsion...) or a field (magnetic or electric field).
– Characteristic is an observable or measurable attribute (length, area, temperature, weight, force...) that can be influenced positively or negatively by an innovation process. The characteristics can be technically contradictory when the obvious solution deteriorates some characteristics and improves others. For example, increasing the strength of a metal plate causing its weight to get heavier. Altschuller defined 39 Characteristics.
– Function expresses the action and the object required to solve an innovation problem such as, for example reducing an object temperature, producing an electric field. The action can be defined by verbs (segment, extract, produce, reduce, increase...), whereas, the object can be a resource or a characteristic.

The user problem is to find the resources (substance, force, field) allowing him to fulfill the required function. Using a TRIZ knowledge base, he can semantically search for examples (or patents) achieving the required function starting from the previous operator groups where:

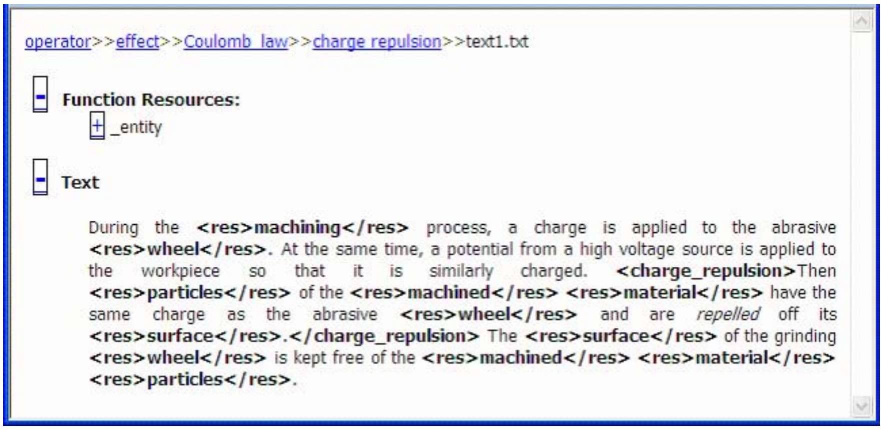
– An inventive principle is defined as an operator which corresponds to a number of contradictory characteristics. Altschuller defined 40 inventive principles. So, the user can creatively resolve a contradiction problem involving different characteristics by the help of several examples (patents).
– A scientific effect is an operator defined by a scientific law of mathematics, physics or chemistry, producing the required action; for example, a force of attraction exists between two charges of different signs, a force of repulsion exists between two similar charges, ("Coulomb's law"). There are several

hundred effects associated to some innovation patents in the implemented TRIZ bases.

– A generic innovating solution is an innovation operator allowing for the improvement or the changing of an action. In this group, the functions and their associated patents are classified by their action: "segment in two parts, segment in several parts, segment in powder...". Altschuller defined 76 generic innovation solutions.

4 Project Objective

In the innovation field, several knowledge bases such as Goldfire Innovator (from Invention Machine)[1], CreaTRIZ[2], Innovative WorkBench IWB IWB[3], TriSolver[4] have already been implemented starting from the innovation method TRIZ (cf. section 3). However the examples suggested are not sufficient to meet the requirement of their users.



operator>>effect>>Coulomb_law>>charge repulsion>>text1.txt

Function Resources:

_entity

Text

During the **<res>machining</res>** process, a charge is applied to the abrasive **<res>wheel</res>**. At the same time, a potential from a high voltage source is applied to the workpiece so that it is similarly charged. **<charge_repulsion>**Then **<res>particles</res>** of the **<res>machined</res>** **<res>material</res>** have the same charge as the abrasive **<res>wheel</res>** and are *repelled* off its **<res>surface</res>.</charge_repulsion>** The **<res>surface</res>** of the grinding **<res>wheel</res>** is kept free of the **<res>machined</res>** **<res>material</res>** **<res>particles</res>**.

Fig. 1. A document and its entities of interest that are annoted by the user interface starting from the knowledge fed into the innovation Base

The objective of the present work is to implement an innovation knowledge base which is independent of the application field and which is auto-generative. In other words, such a base is automatically fed by new examples highlighting the innovation operators and coming from NL texts located on the Web especially on the patent sits. Hence the functions, the resources and/or the characteristics of the operators have to be automatically located in the texts and to be fed into the base. These extracted knowledge entities serve as semantic annotations in

[1] http://www.invention-machine.com/
[2] http://www.knowllence.com/fr/produits/creatriz.php
[3] http://www.ideationtriz.com/new/iwb.asp
[4] http://www.trisolver.com/

their text source where they are dynamically inserted when a user displays it (cf. figure 1). The reason for separation of these annotations from the texts is so that multiple heterogeneous annotations can be made on the same text [14].

5 Automatic Feeding System Architecture

5.1 Working Principle

The working of our system is illustrated in the diagram in figure 2.

Firstly, the system retrieves text documents likely to be, or to contain, operator examples. The research is carried out by an automatic composed query for a conventional research engine such as Google. The keywords necessary for the query composition are extracted from a semantic NetWork (for instance WordNet for English texts). The candidate keywords of the query are those hyponyms of WordNet synsets mapped to innovation concepts (cf. section 5.2).

The structure of the research query is similar to that suggested by [15] in their system associating Web directories to WordNet synsets. For example, let us take

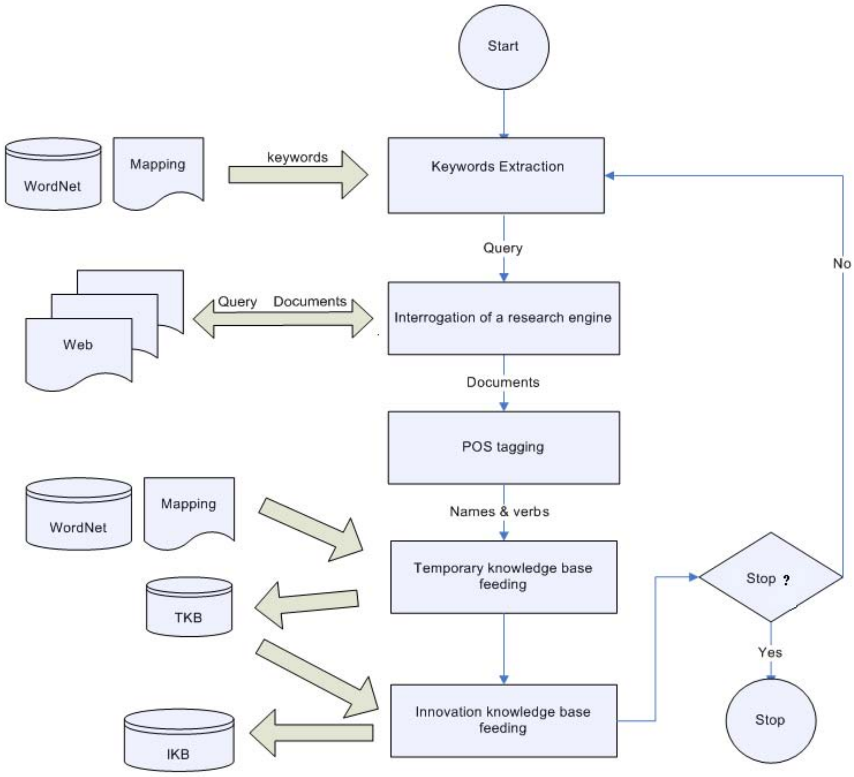


Fig. 2. Working of the system

the query Q = [+circuit "electrical circuit" "electric circuit" "electrical device" -tour -"racing circuit" -lap -circle]. This query makes it possible to seek documents including the words: circuit, electrical circuit, electric circuit, electrical device and which do not include the words turn, racing circuit, lap and circle. So, the returned documents must be relevant to the synsets containing the positive words but not negative ones.

A query makes it possible to recover one or more documents. Each document is separately processed. This process starts by a partial syntactic analysis for recognizing the part of speech (POS: article, noun, verb...) of each word. In the implemented system, it is made by the parser TreeTagger[5].

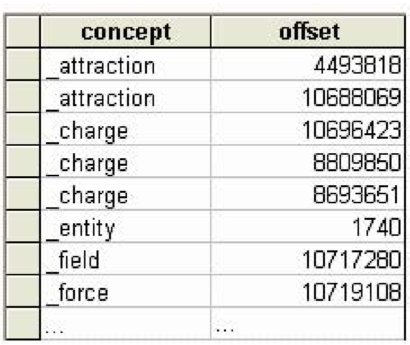
The word, its POS, as well as its associated synsets and their mapped concepts (cf. section 5.2) constitute the primitive candidate knowledge fed into the temporary knowledge base TKB (cf. section 5.3). Then, the relevant knowledge is filtered and fed into the innovation knowledge base IKB (cf. section 5.4) by the specified filtering rules (cf. section 5.5) and the base TKB is emptied.

Lastly, after the filtering process, when finishing all the returned documents for a composed query, a new one can be generated.

5.2 Mapping Between Concepts and WordNet Synsets

Resources, actions and objects of the innovation operators (or their functions) are defined by semantic concepts such as substances, liquid, solid, electric field, attraction force, etc. To locate their instances, which are words or terms in the NL texts, we mapped them to one or more entries a semantic network. This mapping was done to WordNet synsets for the processing of English texts and it was represented by a table consisting of two columns (cf. figure 3) which permit to map a concept name to a synset by its offset. A concept name can be mapped to one or more synset.

For the following, we consider that all the terms associated to the hyponyms of a synset represent the corresponding concept.



concept	offset
_attraction	4493818
_attraction	10688069
_charge	10696423
_charge	8809850
_charge	8693651
_entity	1740
_field	10717280
_force	10719108
...	...

Fig. 3. Mapping table

[5] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html

5.3 Temporary Knowledge Base TKB

This is a base which contains temporary candidate knowledge. This knowledge is also called primitive, because it results from a direct extraction of the NL text sources. It is fed into this base by an automatic system (cf. section 5.1) basing on the mapping and their corresponding terminologies in the semantic network. After that, it is filtered and memorized in the base IKB (cf. section 5.4). The filtering process is carried out by filtering rules (cf. section 5.5).

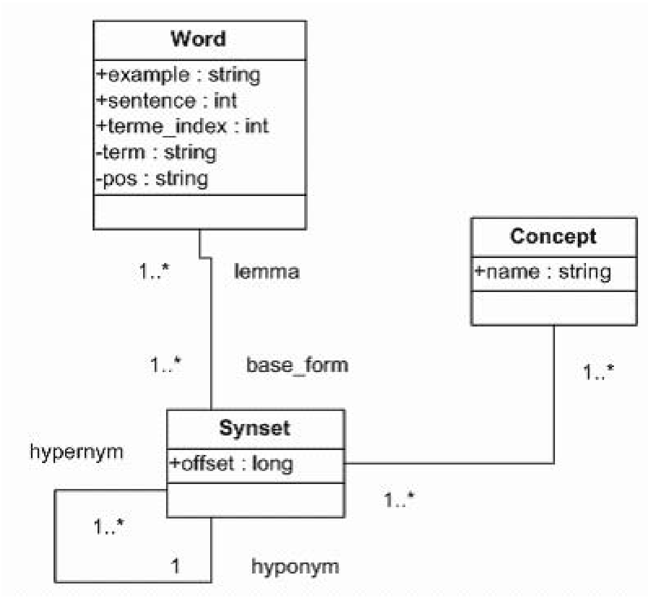
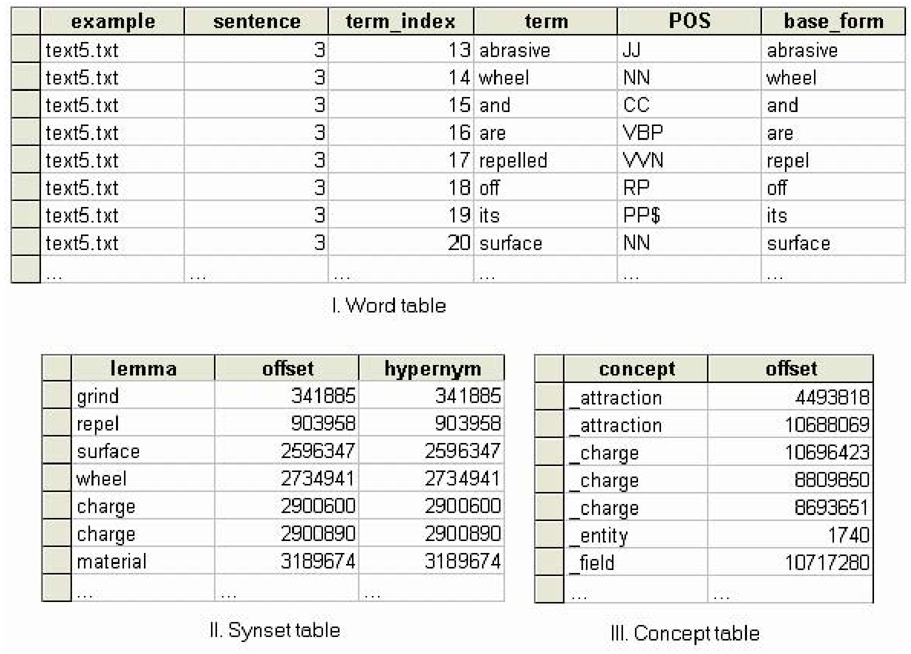


Fig. 4. Conceptual schema of the Temporary Knowledge Base TKB

The conceptual schema of the TKB is given in figure 4. This schema has been implemented with a relational DBMS (cf. section 5.5) and an example of the knowledge data fed into the implemented base is viewed in the figure 5. The principal entities of the diagram are:

– The class Word: an instance of it is a word defined by its text, by the number of its sentences in the text, by its number in the sentence, and by its POS. It is associated to its synsets by its base form called lemma in the WordNet synsets (cf. figure 5.I).
– The class Synset: its instances are the candidate senses of the associated words. These senses are the WordNet synsets defined by their offsets. They are also associated to their relating mapping concepts (cf. figure 5.II).
– The class Concept: its instances are the mapped concepts (cf. figure 5.III).

The feeding process of this base starts from the Word table. The parser annotates the text words by their POS and by their base form. For each word, the



example	sentence	term_index	term	POS	base_form
text5.txt	3	13	abrasive	JJ	abrasive
text5.txt	3	14	wheel	NN	wheel
text5.txt	3	15	and	CC	and
text5.txt	3	16	are	VBP	are
text5.txt	3	17	repelled	VVN	repel
text5.txt	3	18	off	RP	off
text5.txt	3	19	its	PP$	its
text5.txt	3	20	surface	NN	surface
...	...	...	...	...	...

I. Word table

lemma	offset	hypernym
grind	341885	341885
repel	903958	903958
surface	2596347	2596347
wheel	2734941	2734941
charge	2900600	2900600
charge	2900890	2900890
material	3189674	3189674
...	...	...

II. Synset table

concept	offset
_attraction	4493818
_attraction	10688069
_charge	10696423
_charge	8809850
_charge	8693651
_entity	1740
_field	10717280
...	...

III. Concept table

Fig. 5. Example of data fed into the base TKB

text reference, the number of the sentence in the text, the number of the word in the sentence, the word, its POS and its basing form are fed into a row of the table.

Then, the Synset table is fed by the candidate synsets. They are the synsets of the text words, which belong to a hierarchy defined by a concept in the mapping (cf. figure 3). We feed these synsets as follows:

If the POS of the word is noun, for each synset s of its synsets, we recover all the offsets of its hypernym synsets, then we calculate their intersection with the mapped offsets (cf. section 5.2). If the intersection is not empty, we feed in the table 5.II the offset, the lemma of the synset s, and the offsets resulting from the intersection.

For example, let $S(ice) = \{14066911, 8727351, 7145319, 3447964, ...\}$ be the set of offsets of for the synonym synsets of the word ice, let $H(3447964) = \{3447964, 3379566, 3168471, 3647935, 3561924, ...16236, 1740\}$ be the set of hypernyms for the synset having the offset $s = 3447964 \in S(ice)$ and let $M = \{10696423, 10688069, 10688453, 1740\}$ be the set of mapped synset offsets.

So, the intersection $H \cap M = \{1740\} \neq \Phi$. Hence, we feed in the table synset the tuple: $\{ice, 3447964, 740\}$.

If the POS of the text word is verb, adjective or adverb, let S be the whole of the synsets of this word. We carry out the same previous treatment starting from the noun synsets associated to the synsets of S by the Nominalization relation in

WordNet. In the present work, we have ignored the adjectives and the adverbs, because they were generally redundant.

Finally, the Concept table is fed directly starting from the mapping (cf. figure 3).

5.4 Innovation Knowledge Base IKB

The examples illustrating the innovation operators and the associated knowledge to enrich the Innovation Knowledge Base (IKB) are represented by the UML schema in figure 6.

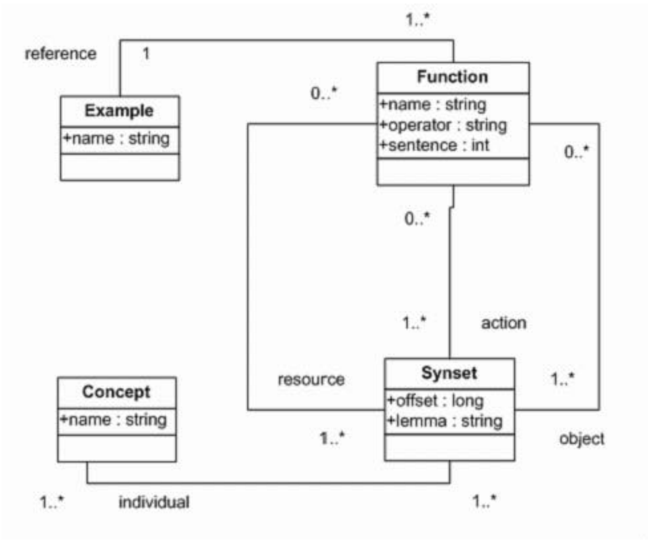
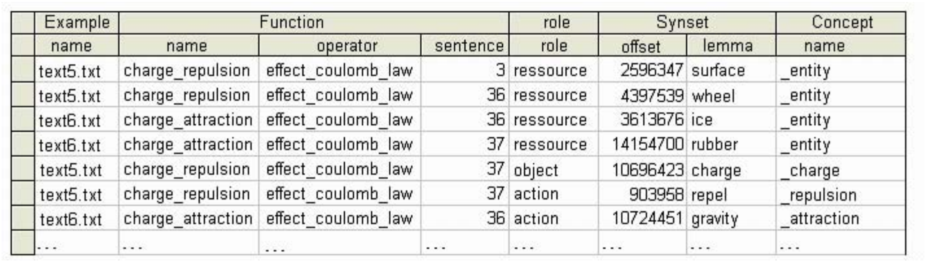


Fig. 6. Conceptual schema of the Innovation knowledge base IKB

The examples are instances of the class Example. They are defined by their text reference. Each one must be associated to a function which is defined by its name, its operator and the number of its corresponding sentences in the example text. Also, a function is associated to its resources, its action and its object by the roles resource, action and object respectively. The domain of these roles is the class Synset. Their instances and those of the class Concept are the instances filtered from the class Synset and the class Concept of the base TKB respectively. For the purpose of giving a semantic interpretation, the association between the two classes is named individual from the said of the class Synset. Therefore, the instances of the class Synset, the synsets, are individuals of the associated instances of the class Concept; the concepts.

This schema is also implemented with a relational DBMS. The contents of the implemented base are fed from the TKB base by filtering rules (cf. section 5.5). The table presented in figure 7 is an example of the knowledge fed into it. Each



Example	Function			role	Synset		Concept
name	name	operator	sentence	role	offset	lemma	name
text5.txt	charge_repulsion	effect_coulomb_law	3	ressource	2596347	surface	_entity
text5.txt	charge_repulsion	effect_coulomb_law	36	ressource	4397539	wheel	_entity
text6.txt	charge_attraction	effect_coulomb_law	36	ressource	3613676	ice	_entity
text6.txt	charge_attraction	effect_coulomb_law	37	ressource	14154700	rubber	_entity
text5.txt	charge_repulsion	effect_coulomb_law	37	object	10696423	charge	_charge
text5.txt	charge_repulsion	effect_coulomb_law	37	action	903958	repel	_repulsion
text6.txt	charge_attraction	effect_coulomb_law	36	action	10724451	gravity	_attraction
...	...	...	...	...	...	...	...

Fig. 7. Example of data fed into the implemented IKB

line represents an association starting with an instance of the class Example, finishing with an instance of the class Concept and passing through each role.

5.5 Feeding the Innovation Knowledge Base

An innovation operator can be associated to one or several functions. A function is characterized by an action, an object, and resources. The objective of the filtering is to identify in a text example these characteristic entities. For that, we considered that all these entities do not necessarily appear in the same sentence. But it is necessary that:

– In a sentence the action is associated to resources;
– In a sentence the object is associated to resources;
– There are common resources between the previous sentences.

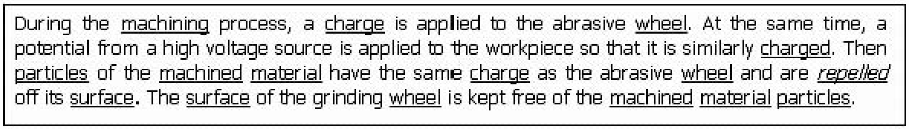
These entities are tracked down by terms. Two terms which possess synsets which are synonym or hyponym/hypernym are considered as equivalents.

Acceptable entities are those that are in correspondence with the predefined mapping (figure 3).

The filtering is operated from the specifications of an expert who has to specify every operator function by one or several triplets (R_i, A, O) where R_i is a resource, A is the action, O is the object. There are as many triplets as resources.

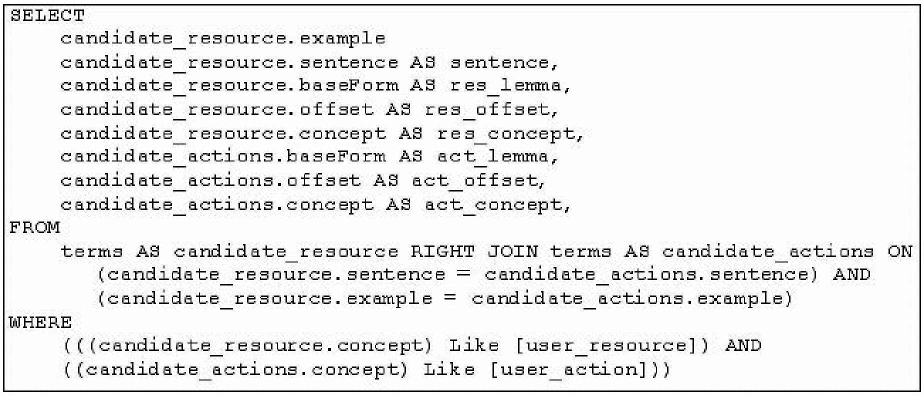
For example let us consider the text in figure 8 illustrating the operator "Coulomb's law" and let us suppose that the expert specified the operator function by the triplet (_entity, _repulsion, _charge)

Acceptable entities according to the predefined mapping are underlined in the text. The action "_repulsion" appears with the term (repelled) and the object "_charge" appears with the term (charge). This example is acceptable because



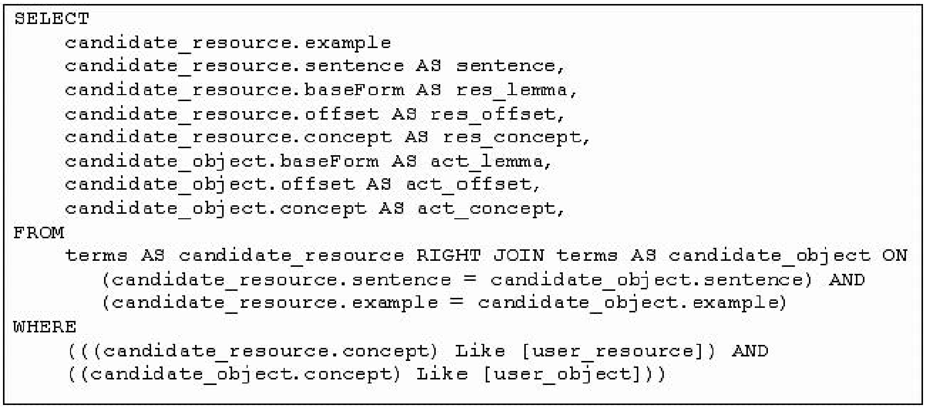
During the machining process, a charge is applied to the abrasive wheel. At the same time, a potential from a high voltage source is applied to the workpiece so that it is similarly charged. Then particles of the machined material have the same charge as the abrasive wheel and are *repelled* off its surface. The surface of the grinding wheel is kept free of the machined material particles.

Fig. 8. An example of a text illustrating Coulomb's law (entities terms are underlined)



```
SELECT
    candidate_resource.example
    candidate_resource.sentence AS sentence,
    candidate_resource.baseForm AS res_lemma,
    candidate_resource.offset AS res_offset,
    candidate_resource.concept AS res_concept,
    candidate_actions.baseForm AS act_lemma,
    candidate_actions.offset AS act_offset,
    candidate_actions.concept AS act_concept,
FROM
    terms AS candidate_resource RIGHT JOIN terms AS candidate_actions ON
       (candidate_resource.sentence = candidate_actions.sentence) AND
       (candidate_resource.example = candidate_actions.example)
WHERE
    (((candidate_resource.concept) Like [user_resource]) AND
    ((candidate_actions.concept) Like [user_action]))
```

Fig. 9. Query filtering the candidate resources and the candidate actions appearing in a same sentence



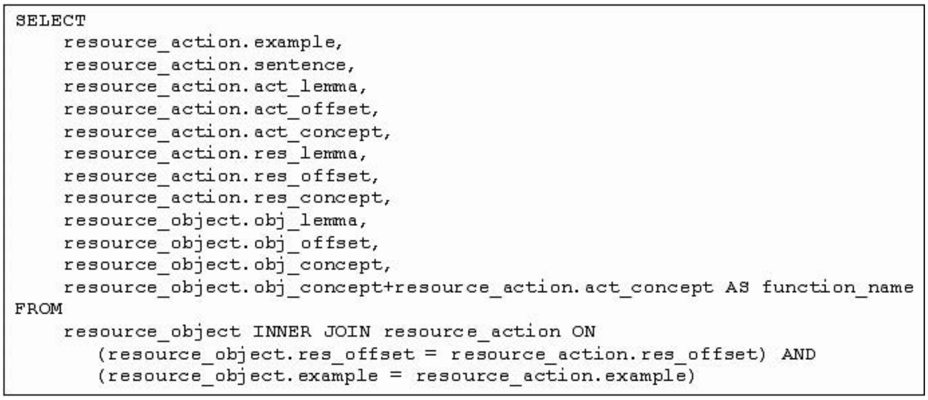
```
SELECT
    candidate_resource.example
    candidate_resource.sentence AS sentence,
    candidate_resource.baseForm AS res_lemma,
    candidate_resource.offset AS res_offset,
    candidate_resource.concept AS res_concept,
    candidate_object.baseForm AS act_lemma,
    candidate_object.offset AS act_offset,
    candidate_object.concept AS act_concept,
FROM
    terms AS candidate_resource RIGHT JOIN terms AS candidate_object ON
       (candidate_resource.sentence = candidate_object.sentence) AND
       (candidate_resource.example = candidate_object.example)
WHERE
    (((candidate_resource.concept) Like [user_resource]) AND
    ((candidate_object.concept) Like [user_object]))
```

Fig. 10. Query filtering the candidate resources and the candidate objects appearing in a same sentence

there are resources (wheel, machine, particle) which appear at the same moment with the action (repel) and with the object (charge) in some text sentences.

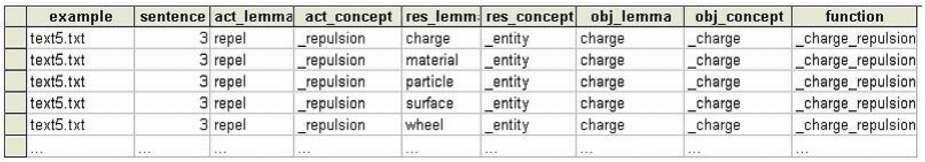
In our relational implementation, the filtering rules are implemented with relational queries. This implementation can be envisaged in a generic way by three queries:

– A first query which looks for associations in a sentence between the resources and the action (figure 9);
– A second query which looks for associations in a sentence between the resources and the object (figure 10);
– A third query which combines the results of the two previous ones by a join on common resources (figure 11).



```
SELECT
    resource_action.example,
    resource_action.sentence,
    resource_action.act_lemma,
    resource_action.act_offset,
    resource_action.act_concept,
    resource_action.res_lemma,
    resource_action.res_offset,
    resource_action.res_concept,
    resource_object.obj_lemma,
    resource_object.obj_offset,
    resource_object.obj_concept,
    resource_object.obj_concept+resource_action.act_concept AS function_name
FROM
    resource_object INNER JOIN resource_action ON
       (resource_object.res_offset = resource_action.res_offset) AND
       (resource_object.example = resource_action.example)
```

Fig. 11. Query filtering the candidate resources and the candidate actions appearing in a same sentence



example	sentence	act_lemma	act_concept	res_lemm	res_concept	obj_lemma	obj_concept	function
text5.txt	3	repel	_repulsion	charge	_entity	charge	_charge	_charge_repulsion
text5.txt	3	repel	_repulsion	material	_entity	charge	_charge	_charge_repulsion
text5.txt	3	repel	_repulsion	particle	_entity	charge	_charge	_charge_repulsion
text5.txt	3	repel	_repulsion	surface	_entity	charge	_charge	_charge_repulsion
text5.txt	3	repel	_repulsion	wheel	_entity	charge	_charge	_charge_repulsion
...	...	...	...	...	...	...	...	...

Fig. 12. Some sentences filtered by the relational queries and their annoted informations

After execution on the extract of the base given in figure 5 one obtains the results of figure 12.

5.6 Displaying the Results

The base IKB can be directly displayed to obtain the functions of an operator and the associated entities.

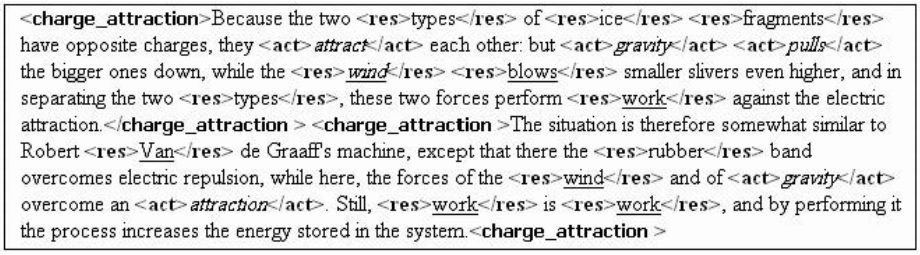
We also foresaw another feature consisting in dynamically inserting annotations resulting from the base during the display of examples. These annotations give the interpretation of terms appearing in the example text (cf. figure 1).

6 Experimentations

Currently, we have specified the functions corresponding to four innovation operators. The approach was tested on a collection of 70 NL documents that are patents extracted from the Web and from the Goldfire Innovator knowledge base using key words corresponding to the specified functions. In this collection, we manually annotated 52 functions and 328 resources in 48 documents in order to compare the results with those obtained from our system. The precision and the recall obtained by our system are presented in the following table (Table 1).

Table 1. Precision and recall obtained by the system

	precision	recall
Document retrieving	0.97	0.93
Function annotation	0.91	0.94
Resource annotation	0.71	0.84



<**charge_attraction**>Because the two <**res**>types</**res**> of <**res**>ice</**res**> <**res**>fragments</**res**> have opposite charges, they <**act**>*attract*</**act**> each other: but <**act**>*gravity*</**act**> <**act**>*pulls*</**act**> the bigger ones down, while the <**res**>*wind*</**res**> <**res**>blows</**res**> smaller slivers even higher, and in separating the two <**res**>types</**res**>, these two forces perform <**res**>work</**res**> against the electric attraction.</**charge_attraction** > <**charge_attraction** >The situation is therefore somewhat similar to Robert <**res**>Van</**res**> de Graaff's machine, except that there the <**res**>rubber</**res**> band overcomes electric repulsion, while here, the forces of the <**res**>wind</**res**> and of <**act**>*gravity*</**act**> overcome an <**act**>*attraction*</**act**>. Still, <**res**>work</**res**> is <**res**>work</**res**>, and by performing it the process increases the energy stored in the system.<**charge_attraction** >

Fig. 13. Example of a relevant text annotation (bad annotations are underlined)

The figure 13 presents an extract of the annotations carried out by the system in a document of the collection. The sentences containing an occurrence of the action and at least one specified resource for the corresponding operator (coulomb's law) are annotated by the operator function name (charge_attraction). The words corresponding to resources and actions are respectively tagged by (res) and (act) in the UI.

The annotation of resources is not as good as that for functions. An important number of resources are ignored and non-resources are retrieved. The precision for resources can be improved by considering extra syntactic and semantic knowledge in specifying the filtering rules or by executing some post-processing in the documents. The recall can be improved too by specifying other operator functions in the innovation domain.

7 Conclusion and Perspectives

In this paper, we have presented an approach and a system for automatically feeding an innovation knowledge base (IKB), starting from texts in natural language. In the base, these texts are considered as examples illustrating their associated innovation operators.

Initially, primitive candidate knowledge is extracted by a POS tagger, guided by mapping towards a semantic network (WordNet). Then, the relevant knowledge is obtained by a semantic filtering process. This process is carried out via rules translating the semantics of the innovation operators. For this purpose, the common knowledge necessary for formulating the operator rules is represented by two conceptual schemas. The first illustrates the primitive candidate knowledge, constituting the filtering rules. The second is the relevant knowledge or innovation knowledge - the object of the rules.

To be significant and visible, the relevant knowledge is dynamically inserted into the original texts when it is displayed or explored through the user interface.

The experiments carried out revealed that our approach is relevant. The remaining annotation errors could be corrected by considering extra knowledge within the queries, as for example, selecting the most appearing resources with the function action and object. Another process can be implemented after the filtering: the grouping of the functions in nominal and verbal sentences in order to select the most syntactically related resource to the action.

In our relational representation there are semantic relations defined as roles such as synonymy and hypernymy/hyponymy. These relations possess the symmetry and transitivity properties. Our relational queries such as we formulated them do not exploit these properties and so our system does not use all the available knowledge. It would be so necessary to evolve towards a more elaborated relational implementation allowing deducing transitive knowledge. Another alternative would be to implement the knowledge bases with a system based on the description logics and to exploit the inference possibilities of this system to make the filtering.

Another problem concerns the specification of the filtering queries. An expert needs to provide the elements of this specification (the semantic representation of innovation operators). It would be interesting to represent these elements in a generic structure from which the filtering queries can be automatically generated.

References

1. Altschuller, G.: Creativity as an Exact Science: The Theory of the Solution of Inventive Problems. Gordon and Breach Science Publishers, New York (1984)
2. Budanitsky, A., Hirst, G.: Evaluating wordnet-based measures of lexical semantic relatedness. Computational Linguistics 32, 13–47 (2006)
3. Choulier, D., Draghici, G.: Triz: une approche de résolution des problèmes d'innovation dans la conception de produits. In: Modélisation de la connaissance pour la conception et la fabrication intégrées, Editura Mirton, pp. 31–58 (2000)
4. Gaizauskas, R., Wilks, Y.: Information extraction: beyond document retrieval. Journal of Documentation 54, 70–105 (1998)
5. Gogu, G.: Méthodologie d'innovation: la résolution des problèmes créatifs. Revue Française de Gestion Industrielle 19, 35–62 (2000)
6. Kem, S.-B., Seo, H.-C., Rim, H.-C.: Information retrieval using word senses: Root sense tagging approach. In: Annual ACM Conference on Research and Development in Information Retrieval, pp. 258–265. ACM Press, New York (2004)
7. Lesk, M.: Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: ACM Special Interest Group for Design of Communication, pp. 24–26 (1986)
8. Linde, H., Hill, B.: Erfolgreich Erfinden - Widerspruchsorientierte Innovationsstrategie, Darmstadt, Hoppenstedt (1993)
9. Maedche, A., Neumann, G., Staab, S.: Bootstrapping an ontology-based information extraction system. Studies In Fuzziness And Soft Computing , 345–359 (2003)
10. Mazur, G.: Theory of inventive problem solving (triz) (1996), http://www.mazur.net/triz/

11. Miller, G.: Wordnet: A lexical database for english. Communications of the ACM 38, 39–41 (1995)
12. Muslea, I.: Extraction patterns for information extraction tasks: A survey. In: AAAI 1999. Workshop on Machine Learning for Information Extraction (1999)
13. Poibeau, T.: L'évaluation des systèmes d'extraction d'information: une expérience sur le français. Langues 2, 110–118 (1999)
14. Popov, A., Kiryakov, D., Ognyanoff, D., Manov, A., Kirilov, M.G.: Towards semantic web information extraction. In: The 2nd International Semantic Web Conference, Florida, USA (2003)
15. Santamaría, C., Gonzalo, J., Verdejo, F.: Automatic association of web directories with word senses. Computacional Linguistics 29, 485–502 (2003)
16. Sickafus, E.N.: Unified Structured Inventive Thinking: How to Invent, Ntelleck (1997)
17. Soderland, S.: Learning information extraction rules for semi-structured and free text. Machine Learning 34, 1–44 (1999)
18. Soderland, S., Fisher, D., Aseltine, J., Lehnert, W.: Crystal: Inducing a conceptual dictionary. In: The 14th International Joint Conference on Artificial Intelligence, pp. 1314–1321 (1995)
19. Terninko, J., Alla, Z., Boris, ZI.: Step-by-Step TRIZ: Creating Innovative Solution Concepts, 3rd edn. Responsible Management Inc., Nottingham (1996)
20. Thompson, C., Califf, M.E., Mooney, R.: Active learning for natural language parsing and information extraction. In: The 16th International Conference on Machine Learning, pp. 406–414 (1999)
21. Wilks, Y., Stevenson, M.: Sense tagging: Semantic tagging with a lexicon. In: Proceedings of the SIGLEX Workshop Tagging Text with Lexical Semantics, pp. 74–78 (1997)
22. Yarowsky, D.: Unsupervised word sense disambiguation rivaling supervised methods. Annual Meeting of the Association for Computational Linguistics , 189–196 (1995)

Ontology Learning for Search Applications

Jon Atle Gulla, Hans Olaf Borch, and Jon Espen Ingvaldsen

Department of Computer and Information Science
Norwegian University of Science and Technology, Trondheim
jag@idi.ntnu.no

Abstract. Ontology learning tools help us build ontologies cheaper by applying sophisticated linguistic and statistical techniques on domain text. For ontologies used in search applications class concepts and hierarchical relationships at the appropriate level of detail are vital to the quality of retrieval. In this paper, we discuss an unsupervised keyphrase extraction system for ontology learning and evaluate its resulting ontology as part of an ontology-driven search application. Our analysis shows that even though the ontology is slightly inferior to manually constructed ontologies, the quality of search is only marginally affected when using the learned ontology. Keyphrase extraction may not be sufficient for ontology learning in general, but is surprisingly effective for ontologies specifically designed for search.

1 Introduction

Traditional ontology engineering approaches are tedious and labor-intensive, as the successful construction of high-quality ontologies requires a wide range of skill sets as well as an ability to deal with very complex and formal representations. The ontologies are expensive to develop and maintain, and it is often hard to manage and coordinate the contributions from various types of domain experts and ontology modelers. The subsea petroleum ontology developed by the Integrated Information Platform project, for example, currently contains more than 55.000 classes, has been constructed on the basis of existing ISO standards over 3 years in a 3 million Euro project and is still not ready as a new ISO standard [10]. At the same time, the ontologies are vital in Semantic Web applications, as they provide the vocabulary for semantic annotation of data and help applications to interoperate and people to collaborate.

Most ontology engineering methods today are based on traditional modeling approaches and stress the systematic manual assessment of the domain and gradual elaboration of model descriptions (e.g. [4,5]).

Ontology learning is the process of automatically or semi-automatically constructing ontologies on the basis of textual domain descriptions. The assumption is that the domain texts reflect the terminology that should go into an ontology, and that appropriate linguistic and statistical methods should be able to extract the appropriate concept candidates and their relationships and properties from these texts. Numerous approaches to ontology learning have been proposed in recent years

[7,10,11,14,15,17], and they seem to allow ontologies to be generated faster and with less costs than manual modeling approaches.

Even though many of the approaches display impressive results, the complexities of ontologies are so fundamental that the generated candidate structures often just constitute a starting point for the manual modeling task. Advanced approaches with deep semantic analyses of text or whole batteries of statistical tests tend to yield better results, but are expensive to develop and may still not compete with traditional ontology modeling with regard to its abilities to represent deep domain properties. However, the real quality of ontologies depends on its use in applications, its *application value*, which necessitates a consideration of how the ontology and the ontology engineering method match the requirements of the application.

Ontology-driven search applications use ontological structures to interpret and reformulate user queries. Only parts of the full ontology is useful to these applications, and the behavior of both the users and the domain collection may affect the way the ontologies should be constructed.

In this paper we present an unsupervised keyphrase extraction system that has been used to speed up the construction of search ontologies. The extracted keyphrases serve as concept candidates in the ontology and can even give indications for how hierarchical relations should be defined. This is a lightweight ontology learning approach, though cheap and practical to use for domains that evolve and lack available domain experts.

The paper is structured as follows. Section 2 discusses the required qualities of ontologies for search. We then introduce the keyphrase extraction system in Section 3 and briefly explain how it compares to manual ontology building based on a real case in Section 4. Section 5 introduces an ontology-driven search engine that uses ontologies to expand user queries. The semi-automatically generated and manually modeled ontologies are both plugged into the search application and evaluated with respect to search relevance in Section 6. Section 7 is devoted to related work, and the conclusions are found in Section 8.

2 Ontological Quality

Ontology-driven information retrieval incorporates a range of features that are not commonly found in traditional search applications. Commercial vector space search engines have started to make use of shallow linguistic technologies, but they do not attempt to expose the underlying semantics of documents [8]. An ontology-driven search application may in principle operate at three different levels of ambition. At the lowest level, we use concept hierarchies in the ontology to retrieve and present ranked documents to the user. The ontology is used to reformulate the query in terms of semantic concepts or to construct semantic indices. Slightly more challenging on the ontology side is the browsing of knowledge in the domain. The idea here is to let users explore relationships and hierarchies in the ontology to help him get an overview of the domain and find related information in a more interactive search session. At the most ambitious level, reasoning is employed to provide answers that are composed of several documents or implied by rules and axioms in the ontology.

A formally defined ontology language like OWL and a complete ontology with constraints and axioms must then be available [1]. Figure 1 illustrates how ontological information may be used in search.

Function	Focus	Ontology specification needed
Retrieve a document	Concepts	Concepts, hierarchies
Browse knowledge	Ontological structures	+ Properties, relationships
Compose a reply	Reasoning	+ logic, constraints

Fig. 1. Three applications of ontological information in information retrieval

As our research on search applications is on pure document retrieval, we will in this paper concentrate on the search quality of ontological concepts and hierarchies. The *ontology value quadrant* in Figure 2 is used to evaluate an ontology's usefulness in a particular application. The ontology's ability to capture the content of the universe of discourse at the appropriate level of granularity and precision and offer the users understandable and correct concepts are important features that are addressed in many ontology/model quality frameworks (e.g. [7,11,15]). But the construction of the ontology also needs to take into account behavioral aspects of the domain as well as the users of the application. For search ontologies, this means that we need to consider the following issues about content and behavior:

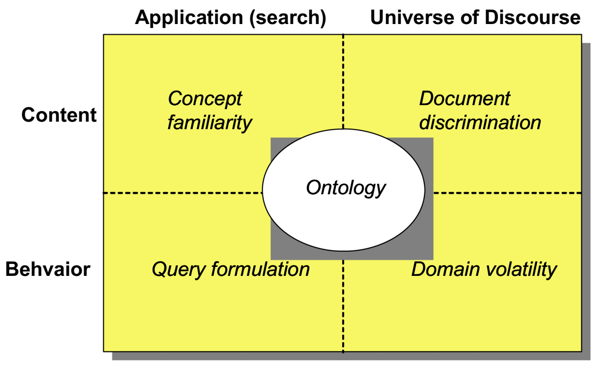


Fig. 2. Ontology value quadrant

- **Concept familiarity.** Terminologies are used to subcategorize phenomena and make semantic distinctions about reality. A high-quality ontology is made up of concepts that correspond to users' way of describing the same phenomena. Analyses of query logs reveal that users tend to use nominal phrases. Whereas we refer to user concepts not found in the ontology as *ignored concepts*, ontology concepts not appealing to users are called *superfluous concepts*.
- **Document discrimination.** The structure of concepts in the ontology decides which groups of documents can theoretically be singled out and returned as

result sets. Similarly, the concepts implied in user queries indicate which groups of documents he might be interested in and which distinctions between documents he considers irrelevant. If the granularity of the user's preferred concepts and the ontology concepts are compatible, combinations of these terms can single out the same result sets from the document collection. Result sets that can be implied by combinations of user-preferred concepts and not by combinations of ontology concepts are called *unfulfilled result sets*. Result sets that can be singled out by combinations of ontology concepts and not by combinations of user-preferred concepts are considered *superfluous result sets.*
- **Query formulation.** The user queries are usually very short, like 2-3 words, and hierarchical terms tend to be added to refine a query [8]. This economy of expression seems more important to users than being allowed to specify detailed and precise user needs, as very few use advanced features to detail their query. Hierarchical ontological structures corresponding to the users' query reformulation strategies are important.
- **Domain volatility**. Both the search domain itself and its documents may be constantly changing, and parts of the domain may be badly described in documents compared to others. The ontology needs to be constructed in such a way that regular and frequent updates are supported.

An ontology learning approach for search ontologies, thus, should be inexpensive and needs to generate familiar candidate concepts that enable the user economically to retrieve exactly those result sets that he might be interested in.

3 Ontology Learning with Unsupervised Keyphrase Extraction

Keyphrase extraction is the process of extracting an optimal set of keyphrases to describe a document. Whereas supervised keyphrase extraction employs a collection of documents with pre-assigned keyphrases to train the extraction algorithm, unsupervised extraction relies solely on a reference collection of plain unannotated textual data. Unsupervised keyphrase extraction has the advantage of being more widely applicable, since the method does not require any knowledge of the domain or consultation of domain experts. On the other hand, supervised keyphrase extraction normally produces more relevant keyphrases and can with repeated training improve the quality of its own keyphrases (see for example [18, 20, 22]).

A list of keyphrases gives a high-level summary of the document content. Such summaries can be used on search engine result pages, helping the user to decide which documents are relevant. It is also often used in document clustering or back-of-book index generation, though in this work we focus on ontology learning. Given a collection of documents describing a domain, the extracted keyphrases can be used to identify important concepts and provide a basis for constructing simple ontologies.

Figure 3 gives an illustration of the various steps in an unsupervised keyphrase extraction system for ontology learning. After cleaning and filtering the domain text, linguistic and statistical techniques are used to extract and rank candidate phrases from the domain. To avoid phrases that are common in all domains, we compare the candidates with a reference text and only include phrases that are characteristic to this particular domain. The final selection is done either by outputting a fixed number of keyphrases from each document in the collection, or selecting all keyphrases scoring higher than some threshold. The phrases may be single words, though usually the most interesting of them are longer noun phrases. After an appropriate set of candidate phrases has been identified, they are verified manually by domain experts and related to each other with various hierarchical and associative relationships. The multi-word keyphrases tend to give useful hints when constructing these hierarchies, but manual work is needed to complete the hierarchies and possibly add more abstract concepts that link everything together in complete ontologies.

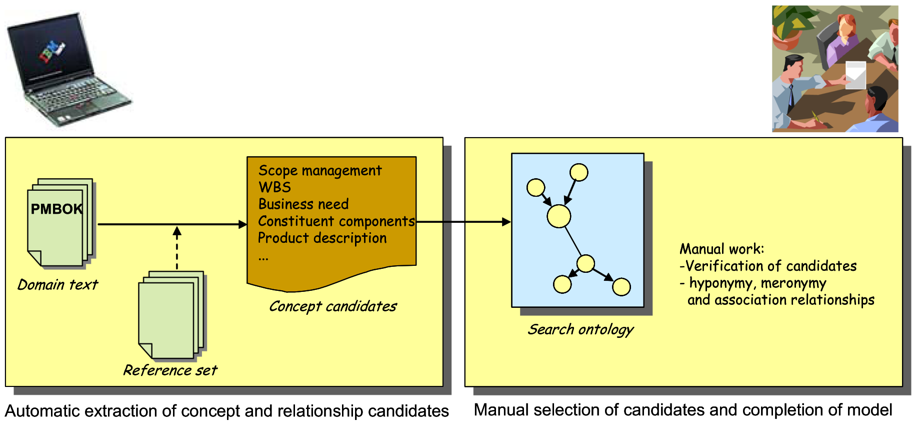


Fig. 3. Ontology learning with keyphrase extraction

4 Building Project Management Ontologies in STATOIL

STATOIL ASA is the leading petroleum company on the Norwegian Continental Shelf and has more than 25,000 employees in 31 countries. Most of their textual documents are structured in NOTES databases, but they are now in the process of implementing new applications and processes for information management. As part of this work, we are building ontologies that can enable ontology-driven search and more efficient application integration.

The domain chosen for the keyphrase extraction system was STATOIL's project management standard, PMI. This standard is enforced throughout STATOIL's organization and is well documented in books and reports. In particular, STATOIL is using a book called PMBOK[1] as a guide to people involved in projects. This book contains 12 chapters that define all the project terminology used in the management of STATOIL projects. We built two independent ontologies of the project management domain, one using keyphrase extraction and one with traditional modeling methods.

[1] Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK), 2000.

Semi-Automatic Ontology Learning

Our unsupervised keyphrase extraction system was first used to extract candidate concepts from PMBOK's 12 chapters. Each chapter in PMBOK was treated as a separate document, and all formatting and document structures were deleted. The resulting input to the extraction system was unannotated plain text, as shown by the PMBOK fragment below:

```
Scope planning is the process of progressively elaborating and
documenting the project work (project scope) that produces the
product of the project.
```

A Brill Part-Of-Speech tagger was then used to tag each word with its respective part of speech (POS):

```
Scope/NNP planning/NN is/VBZ the/DT process/NN of/IN progressively/RB
elaborating/VBG and/CC documenting/VBG the/DT project/NN work/NN (/(
project/NN scope/NN )/) that/WDT produces/VBZ the/DT product/NN of/IN
the/DT project/NN ./.
```

These POS tags come from the Penn Treebank tag set and allow us to filter out words that should not be considered potential keyphrases. Since our keyphrases should be composed of nouns, we concentrated on the words tagged with NN (singular or mass noun), NNP (singular proper noun) and NNS (plural noun). Stopwords were removed from the text, using a list of 571 words that are abundant in the English language and carry little or no discriminating meaning:

Scope planning **is the** process **of** progressively elaborating **and**
documenting **the** project work (project scope) **that** produces **the**
product **of the** project.

The words shown in bold were deleted from the text. To get rid of morpho-syntactic variation in the text, we used a lexicon to lemmatize the words. This means that the actual inflections are replaced by their corresponding base forms, giving us `plan` instead of the progressive `planning` and `produce` instead of the third person singular `produces`. If a word did not occur in the dictionary, Porter's stemming algorithm was applied to the word. This resulted in the following sequence of words (POS tags hidden):

```
Scope plan process progress elaborate document project work project
scope produce product project
```

Notice that the stemming of `progressively` to `progress` makes it appear like a noun, but we kept the tag RB to avoid that `progress` was analyzed as a noun later.

Different extraction systems tend to adopt different strategies for which structures should be considered potential keyphrases. In our system all consecutive nouns were selected as candidate phrases:

```
{scope planning, process, project work, project scope, product,
project}
```

The candidate phrases were weighted using the *tf.idf* measure used in information retrieval. We first calculated the term frequency (*tf*), which gave us an indication of how frequent this phrase was in this chapter compared to other phrases:

$$\mathrm{tf} = \frac{n_i}{\sum_k n_k}$$

where n_i is the number of occurrences of the considered phrase in the chapter, and the denominator is the number of occurrences of all terms (phrases) in the chapter. The total tf.idf score was calculated as shown below and takes into account the distribution of this phrase throughout the document collection:

$$\mathrm{tfidf} = \mathrm{tf} \cdot \log\left(\frac{|D|}{|(d_j \supset t_i)|}\right)$$

where $|D|$ is the total number of chapters in the collection and $|(d_j \supset t_i)|$ is the number of chapters (excluding the current chapter) where the term t_j appears (not equal to 0). The resulting list of weighted phrases were sorted and presented to the user:

```
{(scope planning, 0.0097), (project scope, 0.0047), (product,
0.0043), (project work, 0.0008), (project, 0.0001), (process,
0.0000)}
```

A total of 180 keyphrases, 15 for each chapter of PMBOK, were selected. These were simply the 15 top-ranked phrases of each chapter, based on the tf.idf score. A domain expert from STATOIL was then asked to mark out those keyphrases that would not be suitable as ontological concepts. With these phrases removed, we had 106 phrases left that were manually structured as an ontology. Synonyms were identified, and the appropriate hierarchical relations were added manually to form a full ontology.

The resulting ontology, which was represented in OWL, contained 3 hierarchical levels, 106 concepts (classes) and 6 synonyms.

Manual Ontology Construction

We also constructed a project management ontology manually. The modelers were familiar with the PMI standard in STATOIL, had access to PMBOK, and also had some experience in running small projects using similar methodologies.

The manual modeling process was substantially longer than the semi-automatic ontology learning process. It led to a larger ontology, with deeper structures, more concepts and more synonyms. The manually constructed ontology had 5 hierarchical levels, contained 142 concepts and 26 synonyms.

5 Ontology-Driven Search

For the evaluation of our ontologies' application value, we installed an ontology-driven search application that uses ontologies to interpret and expand user queries. The idea is to add weighted synonyms and semantic relations to retrieve relevant documents that do not necessarily contain the search terms per se. This is an approach that tends to increase recall rather than precision, which is often preferable for such a small domain with a limited number of indexed documents available.

Take for example the query '*human resource*'. The original query terms as well as their synonyms get weight 1.5. Expanding this query with synonyms and semantically related concepts with slightly lower weights, we get the following two reformulated queries for the two ontologies at hand:

Expanded query with manual ontology	Expanded query with automatic ontology
'human resource' (1.5), hr (1.5), 'organizational planning' (1.0), staff (1,0), 'staff acquisition' (1.0), 'team development' (1.0)	*'human resource' (1.5), 'human resource management' (1.0), 'organization chart' (1.0), role (1,0), chart (1.0), staff (1.0), 'staff assignment' (1.0), 'team competencies' (1.0), 'team development' (1.0)*

The term *hr* is given the same weight as *'human resource'*, as it is considered a synonym in this domain. All the other terms are related to 'human resource' through associations or abstractions and are given a weight of 1.0. The different expansions for the two cases reflect the differences of the two ontologies. As seen from this example, the semi-automatically generated ontology has found more semantic links between human resource and other concepts. On the other hand, only the manually built ontology includes the synonym *hr*.

The expanded weighted query is the system's interpretation of the user's real information needs. After mapping the query onto corresponding search terms, a standard vector model based search engine (Lucene) is used to retrieve and rank documents relevant to the new query.

6 Search Quality of Ontologies

The ontologies were first evaluated independently of their applications. Domain experts from STATOIL ranked the ontology concepts with respect to their suitability in a real full-fledged project management ontology. Since the manually constructed ontology was larger, it was not surprising that it also contained more relevant domain concepts. This ontology had 122 very good concepts against the other ontology's 73 very good concepts, which means that that manual process had managed to uncover 67 % more high-quality ontology concepts. If we take the total number of concepts into account, though, the difference of quality is not so great. Whereas in the manual process around 86 % of the concepts were considered to be of high quality, about 69 % of the semi-automatically generated concepts were of the same quality. For an equal number of concepts, thus, we may conclude that the manual process would give us slightly less than 25 % more high-quality concepts.

It could be tempting to improve the semi-automatic ontology by extracting more keyphrases than the 180 we extracted in this experiment. However, it turns out that the quality of keyphrase extraction is highly dependent on the size of documents available to the analysis. As shown in Figure 4, we need documents of at least 5-6,000 words to get an R-precision of more than 0.4 when the top 15 phrases are included, and it seems very difficult for this method to reach 0.6 for even very large documents. For chapters 1, 3, 4, 9 and 10 it would quality-wise have been better to extract fewer than 15 phrases. Chapter 2 does not follow the general trend of getting better phrases with longer documents, as it deals with the context of project management and the extracted terms were considered out of scope by the experts.

A practical evaluation of the ontology-driven search applications was then run on a document index from STATOIL that contained rather small project management and project-related documents. Two separate search applications were set up to work on the same index. Whereas one application used the manually constructed ontology to interpret and reformulate queries, the other one made use of the ontology constructed with the help of our keyphrase extraction system.

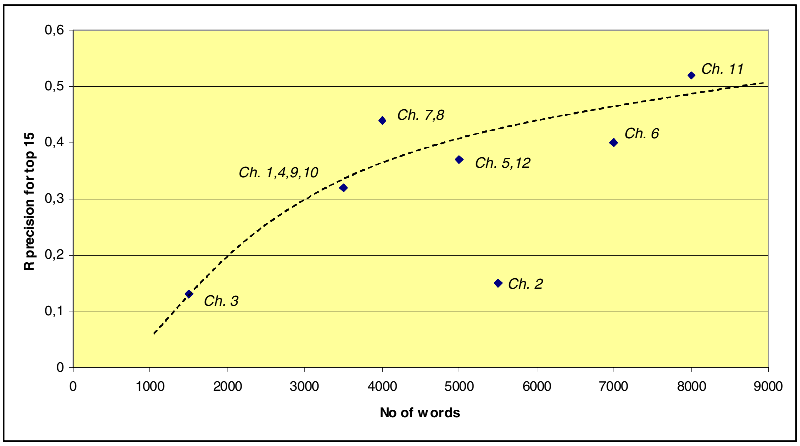


Fig. 4. R-precision and chapter size for extracted ontology concepts

As the intention was to evaluate and compare ontologies rather than to evaluate the search application itself, we defined a total of 16 queries that were all related to concepts in the two ontologies. The expansions of these queries would naturally differ, as the concepts were modeled differently and related to different concepts in the ontologies.

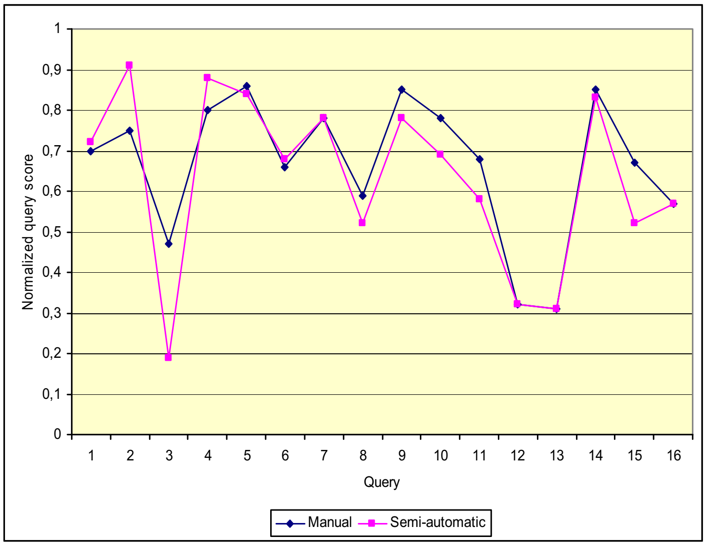
A group of six people were asked independently to run the queries on the two search applications and rate the top 5 documents for each query from 0 (not relevant) to 2 (highly relevant). The total score for one individual's evaluation of one query for one application was given as:

$$Q = 1/2 * \sum_{i=1}^{5} S_i * W_i$$

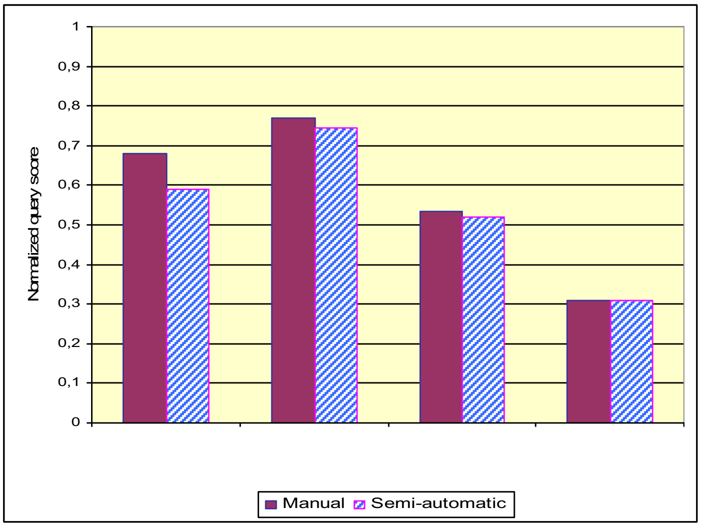
where S_i is the individual rate of document D_i, and W_i is a weight that ranges from 10 for the top ranked document to 1 for the 5th ranked document. After combining the results from each of the six individuals and normalizing the average scores for all queries for the two search applications, we got the results shown in Figure 5(a).

The search application performs slightly better with the manually constructed ontology. In 50% of the queries the manual ontology wins out, and only 25% are answered better with the generated ontology. However, the score differences are in most cases very small. On the average, the query scores for the manual ontology are only 5.1 % higher than for the generated ontology. Taken into account that the manual ontology had 67 % more high-quality concepts and a ratio of good to neutral ontology concepts almost 25 % higher than for the generated ontology, this difference is surprisingly small. It seems that the search quality of ontologies is not so dependent on an exact match between ontological concepts and domain experts' judgments, as long as they are reasonably well defined with respect to the documents available in the domain.

Another interesting observation is illustrated in Figure 5(b). If we group the results on the basis of number of query terms, we can easily see where the two search applications differ in quality. For queries that deal with one-term concepts, like *procurement* and *stakeholder*, the manual ontology performs substantially better than the semi-automatic one. For long detailed queries, like '*cost performance index*' and '*work breakdown structure*', there is practically no difference between the two ontologies.



(a)



(b)

Fig. 5. (a) R-precision for queries. (b) R-precision as function of number of query terms.

This seems to support that the manual ontology contains better concepts and relationships between concepts. Presumably, the importance of good concepts and well-defined relationships are more important when the query is short and vague by itself. For longer queries, the user has already specified his information needs so accurately that the addition of related terms may not contribute much. This may also indicate that ontology-driven query expansion in general has only limited effect when queries are precise and unambiguous.

7 Related Work

In this paper we have investigated to what extent unsupervised keyphrase extraction may be useful in speeding up the construction of ontologies for search applications. Our idea of taking the intended use of the ontologies into account is not new. Thurmair claims that precision and recall are useless in keyphrase extraction, and the quality of extracted terms must be assessed on the basis of how people make use of the terms and how fast they can define their own term subsets [15]. Tomokiyo and Hurst propose an unsupervised extraction strategy based on n-grams, and they require that the users themselves characterize what constitutes proper phrases for their particular applications [17].

One of the most well-known workbenches for ontology learning is Text2Onto, which includes a whole battery of statistical and linguistic text mining components [3]. Text2Onto is meant to support a wide range of analyses and has a flexible and exapandable architecture. This modular approach to text mining is also adopted in other applications [7,10]. As opposed to these workbenches, our system is more lightweight and tailored to the restricted need in constructing and maintaining search ontologies.

OntoLT in Protégé includes traditional statistical methods for term extraction, though its main contribution lies in the use of shallow linguistics to extract structured information from individual sentences [2]. It uses a rule-based system for German and English sentence analysis, SCHUG, to propose properties and relationships based on the recognition of heads, modifiers and predicates in the sentences. A similar approach to linguistic sentence analysis is adopted by Sabou et al. to extract concepts and relationships between concepts in a web service context [14]. These methods are able also to suggest relationships between concepts, but it is an open question how this sentence by sentence approach will work for large text collections where individual sentences are statistically insignificant and aggregated data need to be used to produce representative results.

Our search application had a rather simplistic approach to query expansion. As noted by Voorhees, it is not obvious that adding semantically related terms will improve the quality of the search application [21]. However, experiments with domain-dependent vocabularies – instead of Voorhees' WordNet approach – does indicate that careful semantic refinement of queries may be useful [18]. Mitra et al. [13] is refining the query based on *blind feedback*, i.e. the system itself selects documents that are considered relevant to the original query and uses these documents to construct an expanded query without any human involvement. Similarly, detecting word relationships from result sets and using these to expand the original query with related terms has been tested successfully by for example Xu & Croft [23]. Interestingly, their text mining approach to query expansion has many similarities with our approach using automatically generated ontologies. We apply text mining to construct ontologies off-line, and these ontological structures are afterwards used to expand the queries. A fundamental difference is that our text analysis is done on the whole document collection, whereas their analysis only makes use of documents considered relevant to the unexpanded query.

8 Conclusions

Unsupervised keyphrase extraction is a flexible and inexpensive method for generating candidate concepts to search ontologies. They do not require any particular preparation or involvement of domain experts and are thus well suited to unstable domains like document collections. Using tf.idf to rank keyphrases, we also end up with phrases that are well suited to single out documents in the collection.

The quality of extracted keyphrases is not at the same level as for supervised extraction, though their quality increases with the size of the documents used in the process. It is clear that the keyphrase extraction-based ontology learning method will not produce as many high-quality domain concepts as the manual approach. However, when applied as a search ontology, the quality of the search application is not much affected if a generated ontology replaces a manually built one. For the application value of the search ontology, it seems equally important that the ontology is well adapted to the document collection as that the concepts perfectly model the domain itself. There is a trade-off between the costs of developing and maintaining high-quality ontologies and the benefits of using them in ontology-driven applications.

Unsupervised keyphrase extraction is a promising approach to search ontology engineering, though there are still many aspects of search ontologies that this approach as well as other approaches do not address properly. A good search ontology is specified at a level of granularity that corresponds to the needs expressed in user queries. It should contain concepts that are familiar to the users and allow him to express his information needs in an economic and efficient way. However, we cannot restrict the user to only use already defined concepts and we need a way to interpret user queries that involve non-concept terms that may or may not be related to ontological structures.

References

1. Antoniou, G., Franconi, E., van Harmelen, F.: Introduction to Semantic Web Ontology Languages. In: Eisinger, N., Maluszynski, J. (eds.) Reasoning Web, First International Summer School 2005, ch.1, Springer, Heidelberg (2005)
2. Buitelaar, P., Olejnik, D., Sintek, M.: A Protégé Plug-In for Ontology Extraction from Text Based on Linguistic Analysis. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, Springer, Heidelberg (2004)
3. Cimiano, P., Völker, J.: Text2onto – A Framework for Ontology Learning and Data-Driven Change Discovery. In: Montoyo, A., Muńoz, R., Métais, E. (eds.) NLDB 2005. LNCS, vol. 3513, pp. 227–238. Springer, Heidelberg (2005)

4. Cristiani, M., Cuel, R.: A Survey on Ontology Creation Methodologies. Idea Group Publishing, USA (2005)
5. Fernandez, M., Goméz-Peréz, A., Juristo, N.: Methontology: from ontological art towards ontological engineering. In: AAAI 1997. Proceedings of the Spring Symposium Series on Ontological Engineering, Stanford, pp. 33–40 (1997)
6. Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. Nature Genet. 25, 25–29 (2000)
7. Goméz-Peréz, A.: Evaluation of ontologies. International Journal of Intelligent Systems 16(3), 391–409 (2001)
8. Gulla, J.A., Auran, P.G., Risvik, K.M.: Linguistics in Large-Scale Web Search. In: Andersson, B., Bergholtz, M., Johannesson, P. (eds.) NLDB 2002. LNCS, vol. 2553, pp. 218–222. Springer, Heidelberg (2002)
9. Gulla, J.A., Brasethvik, T., Kaada, H.A: Flexible Workbench for Document Analysis and Text Mining. In: Meziane, F., Métais, E. (eds.) NLDB 2004. LNCS, vol. 3136, pp. 336–347. Springer, Heidelberg (2004)
10. Gulla, J.A., Tomassen, S.L., Strasunskas, D.: Semantic Interoperability in the Norwegian Petroleum Industry. In: Gulla, J.A., Tomassen, S.L., Strasunskas, D. (eds.) ISTA 2006. International Conference on Information Systems and Its Applications (submitted, 2006)
11. Lindland, O.I., Sindre, G., Sølvberg, A.: Understanding Quality in Conceptual Modeling. IEEE Software 11(2), 42–49 (1994)
12. Maedche, A.: Ontology Learning for the Semantic Web. Kluwer Academic Publishers, Dordrecht (2002)
13. Mitra, M., Singhal, A., Buckley, C.: Improving Automatic Query Expansion. In: Proceedings of 21st annual international ACM SIGIR conference on Research and Development in Information Retrieval, pp. 206–214. ACM Press, New York (1998)
14. Navigli, R., Velardi, P.: Learning Domain Ontologies from Document Warehouses and Dedicated Web Sites. Computational Linguistics 30(2), 151–179 (2004)
15. Pinto, H.S., Martins, J.P.: Ontologies: How can They be Built? Knowledge and Information Systems 6(4), 441–464 (2004)
16. Sabou, M., Wroe, C., Goble, C., Stuckenschmidt, H.: Learning Domain Ontologies for Semantic Web Service Descriptions. Accepted for publication in Journal of Web Semantics
17. Thurmair, G.: Making Term Extraction Tools Usable. In: EAMT/CLAW 2003. The Joint Conference of the 8th International Workshop of the European Association for Machine Translation and the 4th Controlled Language Applications Workshop, Dublin (2003)
18. Tomassen, S.L., Gulla, J.A., Strasunskas, D.: Document Space Adapted Ontology: Application in Query Enrichment. In: Kop, C., Fliedl, G., Mayr, H.C., Métais, E. (eds.) NLDB 2006. LNCS, vol. 3999, pp. 46–57. Springer, Heidelberg (2006)
19. Tomokiyo, T., Hurst, M.: A language model approach to keyphrase extraction. In: Dignum, F.P.M. (ed.) ACL 2003. LNCS (LNAI), vol. 2922, Springer, Heidelberg (2004)
20. Turney, P.D.: Learning algorithms for keyphrase extraction. Information Retrieval 2(4), 303–336 (2000)
21. Voorhees, E.M.: Query expansion using lexical-semantic relations. In: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 61–69. Springer, Heidelberg (1994)
22. Witten, I.H., Paynter, G.W., Frank, E., Gutwin, C., Nevill-Manning, C.G.: KEA: Practical automatic keyphrase extraction. In: ACM DL, pp. 254–255. ACM Press, New York (1999)
23. Xu, J., Croft, W.B.: Query Expansion Using Local and Global Document Analysis. In: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 4–11. ACM Press, Zurich (1996)

MultiBeeBrowse – Accessible Browsing on Unstructured Metadata*

Sebastian Ryszard Kruk[1], Adam Gzella[1], Filip Czaja[1,2], Władysław Bultrowicz[1,2], and Ewelina Kruk[1]

[1] Digital Enterprise Research Institute, NUI Galway, Ireland
[2] WETI, Gdansk University of Technology, Poland
firstname.lastname@deri.org

Abstract. Growing abundance of information on the Internet, especially the Next Generation Internet, poses even more challenges on more efficient information management; hence it has brought attention of the researchers to the faceted navigation. Existing solutions, however, do not address the majority of users, who are still inexperienced in using the faceted navigation solutions or who do not understand underlying concepts of the Semantic Web technologies, or both. The query refinement process, while using the faceted navigation interface, is more complex than, e.g., refining a simple keyword-based query.

In this article we present MultiBeeBrowse (MBB), an accessible faceted navigation solution that solves aforementioned problems in the browsing environment. We present how to improve users' access to their history of refinements; we discuss how users can share their browsing experience. And last bust not least, we present an adaptable user interface, which aims to decrease information overload.

1 Introduction

Future Internet, as envisioned by both the academia and the industry, will be the Web of metadata. This large, metadata rich information space, will be hardly manageable with current techniques. Until machines will really understand what we mean, we will have to refine our queries ourselves. The Semantic Web community investigates new solutions for the faceted navigation designed for the unstructured metadata; in contrary to, e.g., Flamenco [18], new interfaces are designed to facilitate browsing without prior knowledge of the structure of the information space.

* This material is based upon works supported by Enterprise Ireland under Grant No. ILP/05/203 and by Science Foundation Ireland Grant No. SFI/02/CE1/I131, and partially by KBN, Poland under Grant No. 4T11C00525. The author would like to acknowledge Daniel Schwabe, Peter Brusilovsky, Bill McDaniel, Henryk Krawczyk, Stefan Decker, the DERI eLearning Cluster, the Corrib.org working group for fruitful discussions, and the group of 20 patient volunteers who agreed to take help with the evaluation.

1.1 Motivations

In the faceted navigation the information space is partitioned using orthogonal conceptual dimensions of the data. These dimensions are called facets and represent important characteristics of the information elements. Each facet has multiple restriction values and the user selects a restriction value to constrain relevant items in the information space. The faceted navigation interface allows to quickly narrow down number of results by choosing more and more precise values from various angles. The facet theory [13] can be directly mapped to navigation in semi-structured RDF data.

The faceted navigation aims to allow users to harness the full potential of the web rich in semantics. Many people, however, have still problems with using keyword search. In 2004, OneStat.com[1] announced that 77% of queries consists of 3, or less, word phrases. In January 2007, RankStat.com[2] reported that 80% of queries consist of 4, or less, word phrases. It means that over last 3 years only a few people started using more complex queries. A query consisting of two word phrases is the most popular, which means that most users do not create complex queries; they rather rely on their luck, reading through long lists of results, and eventually on refining they queries.

Constructing more complex (5 and more words) keyword-based queries still poses problems to an average user; therefore, we cannot expect that advanced navigation, e.g., faceted browsing, as it is delivered at the moment, will gain much attention in the near future.

There are also two other features which still make it easier to construct keyword queries composed to faceted navigation: (1) refining a query by adding, removing, or changing certain words is relatively easy, since we operate in one dimension of vocabulary definitions; we need to remember not only values (keywords) but also the names of the facets (properties). This can become a real nuisance when we trying to refine a query many times, and when by going back in the refining process we loose our previous refinements. (2) we can easily share our keyword-based searching experience with others, by dictating the words, giving a hint on them, or simply copy-pasting the URL from a search engine. The same operation using the faceted navigation is much harder to achieve; it takes much longer, e.g., to dictate all operations a user is required to perform.

In this article we present an accessible browsing solution, a MultiBeeBrowse component for collaborative faceted navigation, which answers aforementioned motivation requirements. By *accessible browsing* we understand a user interaction with the browsing services, where accessibility is achieved through: improving access to the history of refinements, adapting the presentation and browsing style, lowering the information overload, and engaging users in the collaborative browsing. The term *unstructured metadata*, used in this article, adheres to the concept of information, which structure, or schema, has not been defined before, or could not be identified while processing the information.

[1] http://www.onestat.com/html/aboutus_pressbox27.html

[2] http://www.rankstat.com/html/en/seo-news1-most-people-use-2-word-phrases-in-search-engines.html

1.2 Related Projects

With the growth of the Semantic Web we need more powerful tools to search and browse on the unstructured metadata. We have identified four types of the navigation interfaces: (1) with a keyword search, e.g., Swoogle [3], (2) with an explicit querying, e.g., Sesame[4], (3) with a graph visualization, e.g., IsaViz [5], and (4) with faceted navigation, e.g., Longwell[6].

Oren *et al* [12] showed the advantages, such as speed of information retrieval and the intuitiveness, of the faceted navigation. They presented and evaluated, both formally and experimentally, many solutions in this area such as BrowseRDF [12], Flamenco [18], mSpace [14], Ontogator [6], Spectacle[7], and Siderean Seamark[8]. BrowseRDF turned out to be the most complete and powerful in the whole group. Nevertheless the authors omitted one of the best, in out opinion, faceted navigation solutions: SIMILE's Longwell. Another omitted, but very interesting system, is /facet [5] browser which was created and presented at the same time as BrowseRDF.

Longwell is a navigation solution, which builds upon flexibility of the RDF data model and effectiveness of the faceted browsing user interface paradigm. It allows to visualize and browse arbitrary RDF dataset; users can quickly build easy to use web sites, based on their own data.

BrowseRDF is minimalistic in form; Longwell, have a strong arsenal of functions, and at the same time stays user-friendly, looks good and is reasonably fast. It is also quite similar to /facet solution. /facet has most of the Longwell features and a special mechanism to deal with the large number of facets. However, Longwell is more flexible, fully configurable, when displaying the results.

One of research areas, related to the topic presented in this article, focuses on graphical representation of the queries on RDF graphs. Harth *et al.* [4] presented an interesting solution, which could be used to augment existing services, such as BrowseRDF, Longwell, /facet, or MultiBeeBrowse, with SVG-based graphical representation of faceted browsing queries.

Since we lacked an explicit comparison between Longwell and BrowseRDF we decided to evaluate these two solutions together with our own (see Sec. 7). To keep the evaluation easy and clear for users taking part in it we decided not to evaluate /facet as this solution is quite similar to Longwell.

1.3 Outline of the Paper

In the next section we present an overview of browsing operations on interconnected metadata. In section 3 we present how browsing context information can be delivered through a zoomable interface paradigm. Section 4 shows how

[3] http://swoogle.umbc.edu/
[4] http://www.openrdf.org/
[5] http://www.w3.org/2001/11/IsaViz/
[6] http://simile.mit.edu/wiki/Longwell
[7] http://www.aduna-software.com/products/spectacle/
[8] http://www.siderean.com/

search and browsing experience can be exchanged among the users. In section 5 we identify adaptive hypermedia, and other techniques, for improving accessibility of browsing and of presented results. Finally, in section 6 we identify services oriented architecture of our solution, MultiBeeBrowse. Eventually, we report results of our user-based evaluation in section 7.

2 Browsing on Interconnected Metadata

Information discovery is a key capability of modern, information society; growing abundance of information accessible through various media, including the Internet, has always been both the blessing and the curse of the contemporary humankind. When Google showed up, it simple swept away other search engines with its PageRank algorithm. But the new era of the Internet, which grows to become a network of social media and interconnected metadata (semantics), brings new challenges in the field of the information retrieval.

This section presents an overview of different information discovery techniques, especially in the context of semantic and social web.

2.1 From Searching to Filtering

People, at the moment, are getting the grip on the keyword-based search very slowly (see Sec. 1.1); therefore, more complex navigation solutions are still beyond capabilities of the majority of the Internet users. When dealing with metadata information a more advanced search can come handy. In a digital library we can specify that *Kruk* should be matched against names of the authors only, and *navigation* should be found in titles of articles only. A similar feature is offered by Google search; users can restrict their search to a given site only, using a *site:* prefix of the URL of the site. Although more powerful, these advanced search solutions are not very popular and users tend to stick to simple keyword-based queries. Nielsen [11] even suggests to discourage novice users to using advanced search features. Another way to extend search operation is by delivering a natural language query interface. Users can ask complex queries by using natural language sentences [9].

A simple filtering user interface is a step towards better navigation using advanced search capabilities; a user can select a value for each property (facet) from a list of possible values for this property in the given context. This approach seemed to gain more appreciation from the users [15]. It gave incentives to further research in the area of faceted navigation; the Flamenco project [18] is one of most famous results.

2.2 From Filtering to Browsing

The new Web is not just a network of resources and shallow metadata; it is a network of highly interconnected metadata, often called semantics. The new Internet is also a social medium, with contribution from all users. In the Semantic Web, this interconnected information is represented as an RDF Graph [10]. This imposes new challenges on the navigation techniques. It is required not only to

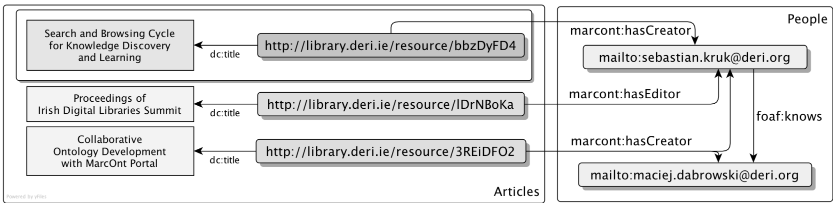

Search and Browsing Cycle for Knowledge Discovery and Learning
dc:title
http://library.deri.ie/resource/bbzDyFD4
Proceedings of Irish Digital Libraries Summit
dc:title
http://library.deri.ie/resource/lDrNBoKa
Collaborative Ontology Development with MarcOnt Portal
dc:title
http://library.deri.ie/resource/3REiDFO2
Articles
People
marcont:hasCreator
mailto:sebastian.kruk@deri.org
marcont:hasEditor
foaf:knows
marcont:hasCreator
mailto:maciej.dabrowski@deri.org

Fig. 1. Example of part of an RDF graph with article and person concepts

filter but also to browse from one set of results (of one type) to a new set of results (of different type).

In our example (see Fig. 1) we have two types of resources: articles and people. A typical scenario for browsing on this information set would be to find other articles written by authors of, e.g., *Search and Browsing Cycle for Knowledge Discovery and Learning*. In our case this could be realized by browsing from this article, to a set of its authors, and browse back to the list of their publications. If we analyze the example graph (see Fig. 1) we will notice that *Proceedings of Irish Digital Library Summit* will not be returned in our browsing process, since the resource is related to the person with *hasEditor* property. In this case, we would need to be able to find similar resources matching values (in this case a person), not properties (*hasCreator*).

This simple scenario illustrates 3 requirements for a browsing on interconnected information:

1. ability to browse from one set of results to another set of results, of possibly even different type, related through a given property;
2. ability to represent, in human readable form, inverse properties (see Def. 1) to those in initial graph of information;
3. ability to find similar resources based on either properties or values related to given set of results.

The first requirement is the basic features of each navigation solution designed to operate on arbitrary information. It allows to find resources related to these in current set. Hence, we can get a list of co-authoring people, based on the initial set of some of their publications.

Definition 1 (Inverse property). *In RDF Graph $G = (V, E, L, l)$, for each $e \in E$, v_1,$v_2 \in V$, label $\in L$, so that e: (v_1, v_2) and l(e) = label, an Inverse Property is a defined as (e_{inv}: v_2, v_1)*[9]*, and $l(e_{inv})$ = inv(label), where inv: $L \rightarrow L'$ is an arbitrary labeling function*[10]

Usability of the aforementioned browsing feature relies on realizing the second requirement - being able to traverse the graph backwards to the direction of given

[9] In most cases the inverse property is a logical concept, which is not explicitly provided in the source RDF graph, but can be explicitly defined using, e.g., `owl:inverseOf`.

[10] Inverse labeling usually involves natural language processing.

property. Support for inverse properties (see Def. 1) relies on both modeling of possible interactions with given set of information, and on delivering user-friendly names for the inverse properties. In our example (see Fig. 1), a user who wants to go from a set of articles to a set of authors, will look for *has creator* property; while, a user who wants to go from a set of people to a set of article, authored by these people, will look for *is creator of* property. In most cases delivering a separate label for the inverse property (in our case *is creator of*) improves the overall usability of the browsing solution (see Sec. 5). The navigation interface should allow for a seamless representation of underlying graph representation of data, without requiring users to understand the technicalities.

The third required browsing operation is finding similar resources based on either given property or on given value. In the first case, it would mean to look for other articles co-authored by the same people who wrote any of the articles from the first set. In the latter case, this would mean to look for any resource somehow[11] related to given value. It would allow for finding publications either authored or edited by given person, but would not restrict the final set of results to the same type of resources as the initial one.

When dealing with resources that can have different properties, such as *has creator* or *has editor*, users might also want to restrict the set of presented results to all resource that have given property, or to those that do not have given property at all.

Some of more complex queries requires couple of steps to build; when you what to say that you are looking for articles that were *presented at a conference* which *took place in Ireland*, and whose *authors* were *members of DERI*, you need to compute queries: (based on conference location and affiliation of authors), and join them together. We have identified 4 types of combination operations:

– **intersection** – each resource must exist in both input sets;
– **sum** – each resource must exist in either of input sets;
– **difference** – each resource must exist in the first set, but not in the second one;
– **binding** – returns a set of results that span together resources from both sets, with a chain of connections not longer than a given parameter.

Access to resource metadata. In most cases, search and browsing services would rather present the result itself; in the context of multipurpose navigation framework, however, access to the resource metadata allows users to build queries on this seed information. They can find similar resources (see Sec. 2.2), or continue with filtering and browsing.

3 Zoomable Browsing Context

Creating a faceted browsing query requires a number of refinement steps (see Lemma 1); each of these steps involves one of atomic operations (see Def. 2) defined in previous section (see Sec. 2).

[11] New results need to have at least one property with a value matching the given one.

Definition 2 (Browsing Operation). *A Browsing Operation $bo \in BO$, is the simplest operation which can be invoked on the navigation engine; it takes output of zero, one, or two other Browsing Operations as input*[12]*; $BO = \{Resource_0, Search_1, Filter_1, Browse_1, Similar_1, Combine_2\}$, each Browsing Operation represents one refinement step in the browsing query building process.*

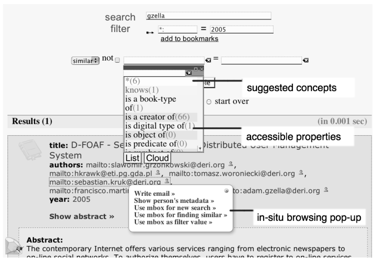
Lemma 1 (Decomposition of Browsing Query). *Each search and browsing query Q can be decomposed into a set of consecutive browsing operations, each representing one refinement step in the query building process.*

Mastering the right query can take some time, especially to novice users; sadly current web browsers does not handle history of actions as a graph of consecutive operations, but rather as a flat list of *recently* opened URLs. Therefore our exact refinement process is lost. When we decided to use backwards button to go back couple of refinements steps, and continue with refinements, our previous history of refinements is only accessibly through a meaningless view of all previous URLs visited with our browser. What if we would like to retrieve our browsing actions from long time ago, and be able to *replay* these refinements, step by step?

Based on these two goals: handling many paths of back and forth refinements, and access to a structured history of operations, we have identified 4 views of browsing context, which allow user to access effortlessly each of aforementioned features.

– *Basic browsing view* provides access to all browsing operations (except for *Combine*) with a typical search and browsing user interface (see Fig. 3). To further enhance usability of our solution, we have extended the query building part with suggestions of properties and values, and results rendering part with an in-situ browsing menu (see Sec. 5).
– *Structured history view* allows users to view their current results in the context of previous and following (if any) operations. This view is almost the same as the *basic browsing view*, with one difference that users see 6 slots (see the *Honeycomb^{TM}view*) with previous operations, and another 6 slots for further browsing. Some of the slots might be already occupied with definition of previously performed refinements; other are free, and a browsing pop-up allows to specify which slot to use to continue browsing (see Fig. 3).
– *Honeycomb^{TM} view* presents users a comprehensive overview of their current browsing context. Each browsing query is represented with a hexagon lozenge in a 3D visualization (see Fig. 3); some edges between hexagons represent browsing operations that were added to the chain of operation to create a new browsing query. With this view, users can get a quick overview on their browsing session; they can quickly jump to a selected browsing query in the current context. This view also allows to perform *Combine* operations, by selecting one of vertexes with exactly two adjacent hexagons. In a similar way, by selecting an edge with only one adjacent hexagon the user can perform any browsing operation, which was allowed in previous views.

[12] Numbers in subscript, next to the name of operation, indicate a number of other browsing operations which can be prepended to given one.



title: D-FOAF - Se… Distributed User Management System
authors: mailto:slawomir.grzonkowski@deri.org, mailto:hkrawk@eti.pg.gda.pl, mailto:tomasz.woroniecki@deri.org, mailto:sebastian.kruk@deri.org, mailto:francisco.marti… …to:adam.gzella@deri.org
year: 2005

Abstract:
The contemporary Internet offers various services ranging from electronic newspapers to

Fig. 2. Basic browsing view

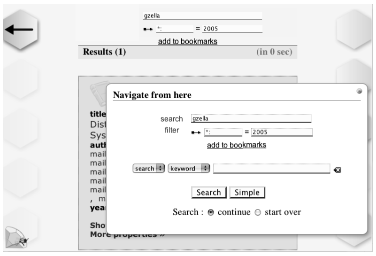


Fig. 3. Structured history view

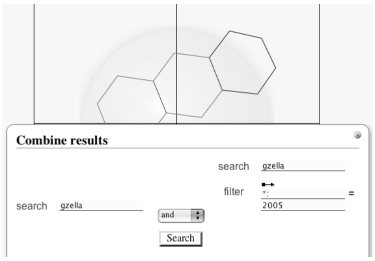
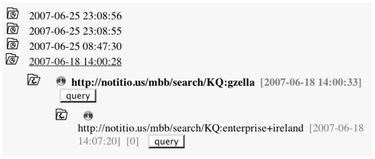


Fig. 4. Honeycomb$^{\mathrm{TM}}$ view



2007-06-25 23:08:56
2007-06-25 23:08:55
2007-06-25 08:47:30
2007-06-18 14:00:28
http://notitio.us/mbb/search/KQ:gzella [2007-06-18 14:00:33]
http://notitio.us/mbb/search/KQ:enterprise+ireland [2007-06-18 14:07:20] [0]

Fig. 5. Life-long history view

– *Life-long history view* presents all previous sessions in which the given query was invoked (see Fig. 3). It allows a user to quickly *move in time* to some browsing context, review refinements invoked after the given browsing query. User can even jump back to that context and continue browsing.

These four views have been set up, so that users can *zoom out* from the view of results, to a view of the browsing context, to a view of all browsing contexts (life-long history).

4 Collaborative Browsing

In the standard keyword-based search queries are not complicated. They usually consist of three or four words (see Sec. 1.1), so it is very easy to recall and repeat searching process. Sharing such queries is also quite easy as it is enough to dictate or send the set of keyword or a URL that invokes the search query. With faceted navigation querying process is not always straight forward. It often consist of keywords and some actions that depends on the previous ones and the context. Thus, sharing queries and results of them is much harder with faceted navigation approach.

Definition 3 (Collaborative browsing). *In group of users U and browsing queries sets B_m, for u_x, $u_y \in U$ and set of browsing queries b_{1_m}, b_{2_m}, ..., $b_{n_m} \in B_{u_m}$, where $m \in (x, y)$. Collaborative browsing is a process in which u_x browse* *through the B_{u_y} and uses $b_i \in B_{u_y}$ Optionally u_x refines query $b_i \rightarrow b_i^{'}$ and performs $B_{u_x} = B_{u_x} \cap b_i$ and/or $B_{u_x} = B_{u_x} \cap b_i^{'}$*

Collaborative browsing (see Def. 3) solution, presented in this paper closes the aforementioned gap between classic keyword-based and faceted browse. All the actions in the system, facets dependency and the context of the search are saved in a special URL (see Sec. 6). By invoking such a URL users can retrace all their browsing steps and get desired results. It is also possible to refine the query if the result is insufficient or the search objective has changed.

Users can now share these specially constructed URLs to the faceted browsing results just like they did with keyword-based search. They can send them by e-mail, through instant messaging, or put it on the web site. But searching for URL in e-mails, browsing instant messaging history, looking for web page is not very convenient and creates lots of problems; user would rather perform new browsing than use the existing one, no matter how good it was. The navigation component can be extended with an online bookmarking solution, such as SSCF (Social Semantic Collaborative Filtering) [7]. SSCF allows users to create the knowledge repository by bookmarking and categorising interesting and valuable resources. Users can utilise the knowledge and expertise of others by browsing and importing resource gathered by their friends. Resources in SSCF are semantically annotated with community established categorization. Users are also able to set fine-grained access rights to each category. SSCF has been extended to deal with URLs representing browsing queries. They became standard SSCF resources, so users can now share browsing experience with ease and without any hassle.

To make the collaborative browsing truly happen, while browsing resources users have quick access to their SSCF bookmarks. They can specify a name and a description and select where in the hierarchy the current browse query be added. It becomes an SSCF bookmark and can be shared by the community. The browsing query has the same features as a standard SSCF resource; it can be copied, cloned, imported by a friend or put in a directory with restricted access rights.

Users can share their browsing queries, and by bookmarking refinements of these queries, they actually perform collaborative browsing. With collaborative browsing people can share their knowledge and their expertise with the social network. Users can also help each other and learn how (by exchanging query bookmarks) to get the satisfying results from the search process.

5 Adaptable Browsing Interface

In this section we will present how adaptive hypermedia, and other personalization techniques, can improve the accessibility measures, such as the speed and the ease of use, of navigation process. An accessible navigation interface contains elements that are automatically adapted by the system according to user's profile, or that can be dynamically changed by users through their actions.

5.1 Results Presentation

As a result of executing the given query, accessible browsing interface returns a set of resources together with all available metadata and RDF relations. A predefined rendering style can be applied based on the value of `rdf:type` property. Each type has some main properties; these properties group predicates describing the same concepts within different ontologies. We have also identified descriptive properties of the most popular resource types, e.g., an *abstract* for an *article*. We use the adaptive hypermedia strechtext technique [1] to present on demand additional, collapsed by default, information, including descriptive properties. A generic result is rendered with only a label, with all other properties collapsed. The label for a generic result, as for all properties and other resources, is generated based on values of `rdfs:label`, `dc:title`, or `foaf:name` properties. Also the URIs of namespaces are abbreviated to short names. This feature adds an aspect of adaptability to the result presentation interface.

5.2 Using Results for Refining Queries

Most of the values displayed in the results page, e.g., property names and values or the resource URIs, can be used for refining user's query. Instead of typing some literals or URIs in search box users can click on the given value and use it for building their query using *in-situ browsing pop-up*; they can start new search or simply load the chosen URI. The action choice box that appears after clicking a value offers the same functionality as the main search box. It limits, however, possible options to only these actions, where the clicked value can be used. It means that users can not load a resource using a literal value or they can not choose a value that is not a predicate for further browsing. System recognizes what element was chosen (property, value, or resource) and adapts the action box by changing the set of possible actions for a given element. This technique is one of the ways of limiting information overload in the browsing solution.

5.3 Adaptive Concepts Suggestion

Whenever users choose an action type and wants to specify some values for this action, such as keywords or predicate name, the list of possible values appears, and they can use one of the suggested entries. A given number indicates the amount of results returned when using this value is presented next to each entry. The list of the results can be presented as a regular list of entries or as a tag-cloud, depending on user preferences.

One of the adaptive aspects of our browsing user interface is suggesting to the user the values while building queries. When browsing through the results or building new queries system suggests the values that user is most likely to use.

The algorithm for suggesting the entries is based on the statistics concerning most popular users choices, most relevant predicates in given context, and the history of browsing saved for a given user. User is not limited to the suggested predicate values; they are only a hint to make the queries building easier. The suggestions appear on the top of the list of possible entries, in different color than the rest of entries. Up to three most likely to be used values are presented.

5.4 Accessible Predicates Names

The faceted browsing user interface is meant to be used by generic users that do not necessary have to be familiar with concepts of RDF. In order to make building of search or browse queries easier we decided to create a kind of "accessible predicates names". At the moment translations of full predicate names and names for inverse properties (see Def. 1) are based on hand crafted list of about 150 items.

Since it is not a very big effort to make such translation for a specific domain, e.g. eLibrary, this could be a good solution even for a final product. Nevertheless we decided to create an algorithm, based on some common predicates naming templates. It transforms the original name to a form that is much closer to natural language. The algorithm defines rules for constructing names of invert properties based on the original name of the predicate.

Transformed predicates are easy to understand and to use even by people without computer background. Transformation process also hides the namespace of each predicate so the users are not confused about its role and meaning.

The verification process of the algorithm, however, is still not finished. Evaluation of the algorithm itself is out of the scope of this paper.

6 Reference Implementation of Accessible Browsing

Based on motivation scenario (see Sec. 1.1), in previous sections we have identified the most important aspects of faceted browsing: a set of basic operations (see Sec. 2), access to context and history of browsing (see Sec. 3), collaborative browsing (see Sec. 4), and adaptability of browsing interface (see Sec. 5).

These component solutions influenced building the MultiBeeBrowse system according to SOA (Service Oriented Architecture) paradigm [3], coupled with AJAX-based user interface. MultiBeeBrowse allows to browse unstructured metadata represented as an RDF graph.

In this section we will present the REST-based services in MultiBeeBrowse SOA, and briefly recap user interface component. We will discuss rationale and main design properties behind this solution.

Why REST solution?. As research [17] shows, SOAP is inadequate for implementing web services for Semantic Web applications. An argument for REST [16], in the context of the MultiBeeBrowse service, is that GET action defines an idempotent request, i.e., subsequent calls of the same URL should return the same results. This can ensure that user will get the same results, each time given URL is called. In MBB it is vital for collaborative browsing paradigm (see Sec. 4), and handling history of results (see Sec. 3).

Our goal was also to construct a meaningful URL representing single browse operations, as well as, whole chain of operations building up a browsing query. This would allow advanced users to quickly construct their queries, directly in the web browser address field.

Overview of MBB services. We have identified following types of services that should be delivered by our SOA:

– *browsing services*: access to a resource, search, filter, browse, similar, combine;
– *context and history management services*;
– *meta-services*, which provides access to statistical information, and allows to format response in desired metadata language format;

Each service is specified using BNF notation[13]; it is enough, since most of the services except for the context services provide only GET method implementation. Each BNF specification has been translated into a regular expression. All services has been grouped in a hierarchical structure. When an HTTP request the URL is processed of the request is decomposed (see Fig. 6) into a chain of atomic service calls, by traversing the tree of hierarchy of services. This approach allows to validate the service call before it is actually executed. Plus, it adheres to web approach to deliver as much as it is possible, against existing problems. In the latter case, if one of atomic browse operations is misspelled the service is executed until the last correct one.

Browsing services specified in section 2, deliver the primary functionality of the MultiBeeBrowse component.

– *Access resource service* allows to load metadata about a resource with given URL for further browsing; this service can only be called as the first one in the chain of browsing operations.
– *Search services*, with three implementations: *keywords and advanced search*, *natural language query templates* [9], and *direct RDF query service.* The two first ones were already described in section 2.1; the third one allows to specify query in one of RDF query languages.
– *Filter service* allows to specify selection filter using either predefined names of properties or using full URIs for both properties and values.
– *Similar service* allows to find resource similar to those in given set,
– *Related service* allows to find resource that are related to the given ones with given property, e.g., (list of publications) → `has creator` → (list of co-authors).
– *Combination service* performs four operations: conjunction, sum, difference, binding, on two given sets of results; it has to be called as the first one in the chain, and takes as a parameters IDs (see *Meta-services*) of two other browsing operations.

Context and history management services. The information on the context of browsing is kept in the RDF storage according to a simple ontology. This ontology defines a *Browsing Context* as a set of current browsing queries (*Browse Action*). Each browsing query invokes a *Call* to a MBB service. The distinction between browsing queries and calls to MBB is important to keep track of

[13] http://wiki.s3b.corrib.org/MBB/SOA

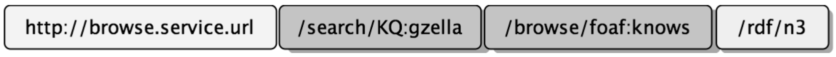


Fig. 6. Example of decomposing browsing query into browsing and meta-service

user actions, through *Browse Action*, and still have single *Calls* independent of the user; this can be used for, e.g., caching purposes later. With context services, users can traverse current browsing contexts, or retrieve all their browsing contexts with the given call.

Meta-services. Meta-services allow to render results in one of RDF serializations (*GetRawMetadata*), or in one of feed (RSS, Atom) formats (*GetFeedMetadata*). For adaptable user interface purposes a special meta-service (*GetStatsMetadata*) generates statistics on properties and values a user can select from. Similar meta-service generates a list of most frequently used concepts (*GetFreqConcepts*). Another meta-service, *GetChainMetadata* renders the definition of current chain of browsing operations in HTML or XML. Finally, the *GetId* service generates a unique ID for given browsing query.

7 Evaluation

Since the faceted navigation itself is pretty well evaluated concept [12] we decided to compare our prototype with other somehow similar solutions such as BrowseRDF[14] and Longwell. Second part of our evaluation is dedicated to gain some opinions about each particular solution delivered by MultiBeeBrowse; these solutions did not exist in either of compared systems (BrowseRDF, Longwell).

Initial questions. Our survey involved 20 people. Distributions of age and education level are shown on the Fig. 7 and Fig. 8. No one way familiar with either the dataset used in the evaluation or with MultiBeeBrowse.

First of we asked couple of questions about subjects' background and knowledge of the area of our research. Slightly more than a half of (60%) have computer science background but only 10 of them knew what RDF is. 15 (75%) of subjects were **not** familiar with the term "faceted navigation", and more than 85% use Google to search information on the Web.

We asked two open, general questions. First was to name some typical tasks they perform during searching. Many (75%) subjects pointed that they simply try to assort good keywords. Then they look on couple first results (30%) or browse the results further (25%). Opening of the results in new browser tabs is also quite popular (20%). Among answers mentioned above we identified: bookmarking, filtering and finding the queries and access to multilevel history of browsing. Second task was to name features subjects miss searching systems they use. Apparently only 20% is fully satisfied - 4 people answered "nothing". In general subjects complained about too many irrelevant results (low precision),

[14] http://browserdf.org

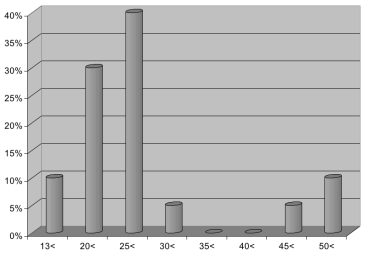


Fig. 7. Distribution of age

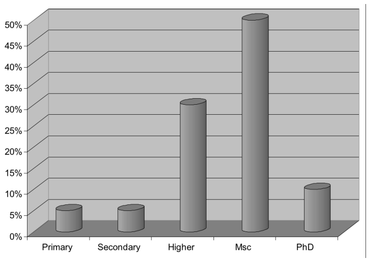


Fig. 8. Distribution of education

lack of quick previews of the results, no possibility of querying in natural language, lack of good history and way to save the results, and last but not least, about problems with too complicated usage and lack of easy help.

We have asked additional pre-assessment questions. And they were (number of positive answers in brackets):

– Have you ever wished you could share your browsing experience with your friends? (45%)
– Have you ever wished you could have someone else help you with browsing? (80%)
– Have you ever wished you could recall you previous search/browse/refinements history? (85%)

Comparison. We gave a short tutorial showing features of every search system as best as it is possible. All of them were working on the same dataset, taken from http://notitio.us service, filled with FOAF [2] profiles of the users, information about publications from JeromeDL [8] (http://library.deri.ie) and an informal knowledge gathered by IKHarvester[15]. Subjects were allowed to ask questions and try to play with each system before the evaluation.

After the presentation, we asked everyone to give a subjective marks (1-10) to each of the compared systems, based on how it looks, how it interacts with the user, features and results it provides.

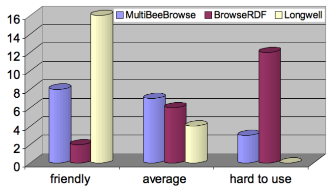
Longwell took first place with average mark 6.875, second (by a proverbial hair's breadth) was our MBB with average 6.8, and left BrowseRDF far behind (avg. 3.8). Since Longwell is a very mature product we found these results satisfying.

We also asked about user friendliness (see Fig. 9), and MultiBeeBrowse came close to Longwell again, leaving BrowseRDF behind.

MBB features. In this paragraph we want to show how users value experience our system. Figures 10 and 11 present marks which were given by the users to particular solutions and to parts of our prototype (all marks are in percents).

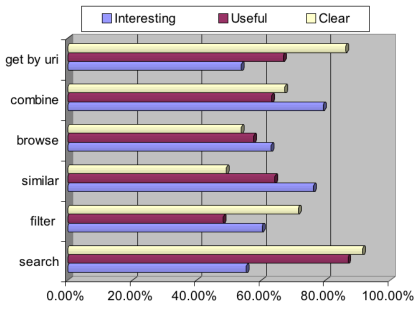
We also asked some open questions, and the results are very good for us. 18 subjects (90%), like the idea of using results for refining queries. The same

[15] http://notitio.us/ikh/



	friendly	average	hard to use
MBB	8	7	3
BrowseRDF	2	6	12
Longwell	16	4	0

Fig. 9. Comparison of user friendliness



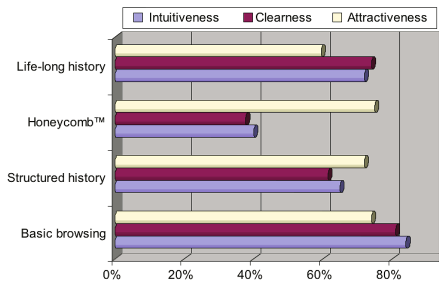
Type of query	Interesting	Useful	Clear
get by uri	53.8%	66.9%	86.2%
combine	79.2%	63.3%	67.3%
browse	63.1%	57.7%	53.8%
similar	76.2%	64.2%	49.2%
filter	60.3%	48.2%	71.5%
search	55.4%	86.9%	91.5%

Fig. 10. Evaluation of accessibility of browse services

number like the idea of saving queries as bookmarks. 15 subjects (75%) can imagine using their friends bookmarked queries for their own searches and vice versa. In particular: while searching for resources (11/55%), helping friends with constructing queries (5/25%) or simply sharing knowledge (12/60%).

As far as user friendliness goes 19 people (95%!) said that the type of the search result was to easy identify, and 17 (85%) of them thought that the default information describing each resource was sufficient. 14 subjects (70%) claimed that the history of searching was easy to identify and 13 (65%) were able to locate themselves while browsing the history. 18 people (90%) found it easy to switch between system components/views.

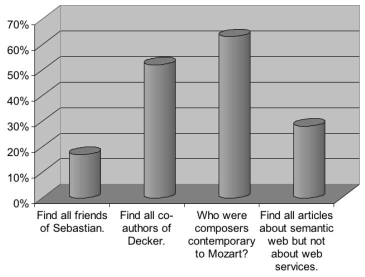
To finish our survey we gave four tasks/questions to each subject. Solutions to similar tasks with use of all three tools were presented during our short tutorial. The tasks were to find: (1) all friends of Sebastian; (2) all co-authors of Decker; (3) who were composers contemporary to Mozart? (4) all articles about semantic web but not about web services. Subjects were asked about their "level of confusion" while the time and the number of the operations needed were measured. Figure 12 shows the average results (the "ideal" results in brackets). By "ideal" we mean the best result achieved during our pre-survey trials. An average number of operations was pretty close to ideal so the system is not really hard to learn and use and the time was approximately twice longer probably due to lack of user experience.

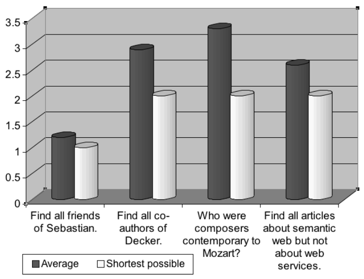


Intuitiveness	Clearness	Attractiveness
72%	74%	59.5%
40%	37.75%	75%
65%	61.5%	72%
84%	81%	74%

Fig. 11. Evaluation of MBB context zooming views

Task	Confusion	Operations	Time
1	17%	1.2 (1)	24s (10s)
2	52%	2.9 (2)	66s (30s)
3	63%	3.3 (2)	106s (30s)
4	28%	2.6 (2)	51s (20s)





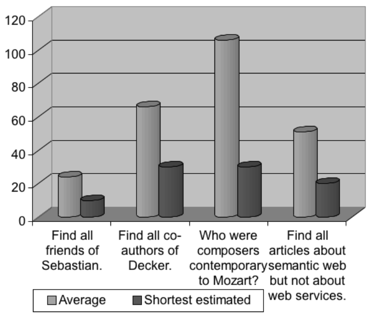


Fig. 12. Evaluation of MBB while performing predefined tasks (confusion, number of operations, time of executing each task)

Table 1. Comparison of browse operations supported by different solutions

Operator	MMB	BrowseRDF	Longwell	Flamenco,mSpace,Ontogator,Spectacle,Seamark
search	+	±	±	-
selection	+	+	±	±
property	+	±	-	-
browse	+	-	-	-
combine	+	±	±	±

Comparison between MultiBeeBrowse and other navigation solutions. We have also compared features provided by MBB, Longwell, BrowseRDF, and other faceted navigation solutions. From the plethora of services provided by MBB (see Tab. 1), only selection services were fully supported by BrowseRDF, and similarity by Longwell.

8 Conclusions and Future Work

In this article we have presented a concept of an adaptable collaborative browsing interface. We have analyzed operations expected from a browsing solution on interconnected metadata (semantic). We have exemplified how context information can support a history of browsing actions; than we have presented the idea of collaborative browsing. Finally, we have described adaptive solutions for the user interface, and presented the REST-based SOA, implemented in our prototype called MultiBeeBrowse. We have also analyzed evaluation results.

MultiBeeBrowse component has been successfully implemented in notitio.us; it is a service for collaborative knowledge aggregation and sharing; it employs IKHarvester for retrieving RDF information about web resources bookmarked by the users. In contrary to other bookmarking services, such as del.icio.us, notitio.us keeps rich, semantically interconnected metadata shared by the users using Social Semantic Collaborative Filtering [7]. MultiBeeBrowse has also been integrated into JeromeDL 2.1, where together with solutions like Exhibit and TagsTreeMaps it enhances user browsing experience.

During our work, and further evaluation of MultiBeeBrowse, we have identified a number of goal for future development of the project. These include AJAX proxy service, for aggregating large number of service calls from the user interface.

References

1. Brusilovsky, P.: Methods and techniques of adaptive hypermedia. User Modeling and User Adapted Interaction 6(2-3), 87–129 (1996)
2. Dodds, L.: An Introduction to FOAF (2004), `http://www.xml.com/pub/a/2004/02/04/foaf.html`
3. Erl, T.: Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Prentice Hall Professional Technical Reference (2004)
4. Harth, A., Kruk, S.R., Decker, S.: Graphical representation of rdf queries. In: Carr, L., Roure, D.D., Iyengar, A., Goble, C.A., Dahlin, M. (eds.) WWW 2006. Proceedings of the 15th international conference on World Wide Web, pp. 859–860. ACM Press, New York (2006)
5. Hildebrand, M., van Ossenbruggen, J., Hardman, L.: /facet: A browser for heterogeneous semantic web repositories. In: International Semantic Web Conference, pp. 272–285 (2006)
6. Hyvonen, E., Saarela, S., Viljanen, K.: Ontogator: Combining view- and ontology-based search with semantic browsing. In: Proc. of XML (2003)

7. Kruk, S.R., Decker, S.: Social Semantic Collaborative Filtering with FOAFRealm. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, Springer, Heidelberg (2005)
8. Kruk, S.R., Decker, S., Zieborak, L.: JeromeDL - Adding Semantic Web Technologies to Digital Libraries. In: Andersen, K.V., Debenham, J., Wagner, R. (eds.) DEXA 2005. LNCS, vol. 3588, Springer, Heidelberg (2005)
9. Kruk, S.R., Samp, K., O'Nuallain, C., Davis, B., McDaniel, B., Grzonkowski, S.: Search interface based on natural language query templates. In: Proceedings of IADIS International Conference WWW/Internet 2006 (2006)
10. Manola, F., Miller, E.: RDF primer. W3C Recommendation (2003)
11. Nielsen, J.: Designing Web Usability: The Practice of Simplicity. New Readers Publishing, Indianapolis (2001)
12. Oren, E., Delbru, R., Decker, S.: Extending faceted navigation for RDF data. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (eds.) ISWC 2006. LNCS, vol. 4273, Springer, Heidelberg (2006)
13. Ranganathan, S.R.: Hidden roots of classification. Information Storage and Retrieval 3(4), 399–410 (1967)
14. Schraefel, M., Wilson, M., Russell, A., Smith, D.A.: mspace: Improving information access to multimedia domains with multimodal exploratory search. Communications of the ACM 49(4) (2006)
15. Sinha, V., Karger, D.R.: Magnet: supporting navigation in semistructured data environments. In: SIGMOD 2005. Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 97–106. ACM Press, New York (2005)
16. Vinoski, S.: Putting the "web" into web services. IEEE Internet Computing 6(4), 90–92 (2002)
17. Wozniak, M.: Service oriented architecture for collaboration and negotiation ontology management portal. Master's thesis, Gdansk Univeristy of Technology, Poland (2006)
18. Yee, P., Swearingen, K., Li, K., Hearst, M.: Faceted metadata for image search and browsing. In: Proceedings of ACM CHI 2003 (2003)

Matching of Ontologies with XML Schemas Using a Generic Metamodel

Christoph Quix, David Kensche, and Xiang Li

RWTH Aachen University, Informatik 5 (Information Systems), 52056 Aachen, Germany
{quix,kensche,lixiang}@cs.rwth-aachen.de

Abstract. Schema matching is the task of automatically computing correspondences between schema elements. A multitude of schema matching approaches exists for various scenarios using syntactic, semantic, or instance information. The schema matching problem is aggravated by the fact that models to be matched are often represented in different modeling languages, e.g. OWL, XML Schema, or SQL DDL. Consequently, besides being able to match models in the same metamodel, a schema matching tool must be able to compute reasonable results when matching models in heterogeneous modeling languages. Therefore, we developed a matching component as a part of our model management system *GeRoMeSuite* which is based on our generic metamodel *GeRoMe*. As *GeRoMe* provides a unified representation of models, the matcher is able to match models represented in different languages with each other. In this paper, we will show in particular the results for matching XML Schemas with OWL ontologies as it is often required for the semantic annotation of existing XML data sources.

GeRoMeSuite allows for flexible configuration of the matching system; various matching algorithms for element and structure level matching are provided and can be combined freely using different ways of aggregation and filtering in order to define new matching strategies. This makes the matcher highly configurable and extensible. We evaluated our system with several pairs of XML Schemas and OWL ontologies and compared the performance with results from other systems. The results are considerably better which shows that a matching system based on a generic metamodel is favorable for heterogeneous matching tasks.

1 Introduction

Integration of information systems is a major challenge that has been addressed in several disciplines such as database and semantic web research. One of the key issues in integration is creating a mapping between the data models of the systems involved. This work is, for example, required if the data from different data sources must be merged in a data warehouse or if two e-business systems must communicate with each other.

Schema matching is the task of identifying a set of correspondences (also called a morphism or a mapping) between schema elements. Many aspects have to be considered during the process of matching, such as data values, element names, constraint information, structure information, domain knowledge, cardinality relationships, and so on. All this information is useful in understanding the semantics of a schema, but it can be a very time consuming problem to collect this information. Therefore, automatic methods are required for schema matching.

A multitude of methods have been proposed for schema matching [23,25] using different types of information to identify elements or focusing on models represented in a specific modeling language such as the Relational Data Model, XML Schema, or OWL [5,7,8,16,17]. To avoid confusion with the terms being used in this paper (e.g. metamodel, model, schema), we want to clarify the terminology first. We will use the terminology defined in the IRDS standard [9] and used by the Object Management Group [21]. According to the IRDS standard, models, schemas, and ontologies are all on the same level and describe the structure of data instances. A metamodel (or a modeling language) is used to define a model, schema or ontology. Examples for metamodels are OWL, UML, or the Relational Data Model.

The schema matching problem is aggravated by the fact that models employed by one system are often represented in different modeling languages. Consequently, besides being able to match models in the same metamodel, a schema matching tool must be able to compute reasonable results when matching models in heterogeneous modeling languages. This is for example required for the annotation of existing XML or relational data sources with ontologies, to enable semantic queries to these sources. Another example is the enrichment of XML web services with semantic information to get semantic web services.

In this paper, we present the matching system which is part of our generic model management system *GeRoMeSuite* [13,14,22]. *GeRoMeSuite* is based on our role-based metamodel *GeRoMe* ([12], phonetic transcription: dʒeɪrəʊm) which provides a generic but yet detailed representation of models represented in different modeling languages. By using *GeRoMe*, our system is able to match models expressed in heterogeneous modeling languages which we will apply in this paper to the case of matching XML schemas with OWL ontologies.

Currently, schema matching systems represent models as directed labeled graphs to support the matching of models from different metamodels. However, the way how a model is encoded as graph is crucial for the match result as structural similarities are also important in schema matching. As models from different metamodels are represented differently in graphs (different labels, different structures), the matching between such models produces often poor results. As we will show, *GeRoMeSuite* produces significantly better results for matching models from heterogeneous metamodels which indicates an advantage of using a generic metamodel for the representation of models. We evaluated the matching performance of our system using various examples for matching OWL ontologies with XML schemas.

The main contributions of our work are (i) a system for matching models using a true generic representation, (ii) which provides several matchers and traversal strategies, and (iii) is based on a very flexible and easily extensible implementation. The generic representation of schemas allows us to apply our implementations of matching algorithms to any combination of models regardless of the modeling languages that the models are originally represented in. Furthermore, the high level of detail of our generic representation enables us to provide different structural views on a model to structure-level matching algorithms. In doing so structural matchers can, for instance, incorporate into their similarity assessment associations between types or derivations between types (the IsA-hierarchy) or even both.

The structure of the paper is as follows. In the next section, we will discuss existing approaches to schema matching. Then, we will describe briefly our generic metamodel *GeRoMe* using an ontology and an XML schema as example. Section 4 presents the system we have developed in terms of its architecture and implemented matchers. In section 5, we present the evaluation of our system. We also discuss and analyze the results of the tested schema matching systems. We conclude our paper with a discussion of our approach and an outlook to future work.

2 Related Work

There have been many approaches to schema matching. The main reason for the various approaches is that each matching problem has its own characteristics and might require a specific solution. The approaches to schema matching can be distinguished by the information they use: some focus only on the schemas, some use external information in form of thesauri, dictionaries or acronym databases, and, if available, it is also possible to use the instance data to find similarities between schemas [23,25].

The Cupid algorithm [17] is intended to be generic across data models and has been applied to XML and relational examples. It uses auxiliary information sources for synonyms, abbreviations, and acronyms. It implements a generic schema matching algorithm combining linguistic and structural schema matching techniques. The input schemas are encoded as graphs. Nodes represent schema elements and are traversed in a combined bottom-up and top-down manner. The matching algorithm consists of three phases. The first phase (linguistic matching) computes linguistic similarity coefficients between schema element names (labels). The second phase (structural matching) computes structural similarity coefficients which measure the similarity between contexts in which individual schema elements occur. The main idea behind the structural matching algorithm is to propagate the similarity of leaf items to the similarity of inner nodes. Finally, the third phase (mapping generation) computes weighted similarity coefficients and generates final mappings by choosing pairs of schema elements with weighted similarity coefficients which are higher than a threshold.

A similar idea is followed by the Similarity Flooding algorithm [19]. Schemas are also represented as directed labeled graphs. Based on the idea that if two nodes are similar then also their neighbors are similar, the similarity of two nodes in the graph is propagated to its neighbors. This procedure is repeated until the Euclidean distance between two subsequent similarity matrices is below a certain threshold. The initial input similarities can be computed by any kind of (linguistic) matching method. The algorithm can be applied to arbitrary graph structures. In [19], there are also several strategies proposed to filter the mapping pairs from the computed similarity values.

The COMA schema matching system is a platform designed to combine multiple matchers in a flexible way [5]. It provides a large number of individual matchers, which contains both terminology approaches and structural approaches. After combining the mapping results from the individual matchers, the output mapping could be chosen as the final result or reused as an individual matching result. As a generic matching system, COMA accepts different schema types as input, such as XML schemas and relational schemas, which are internally represented as directed graphs. COMA also allows users to reuse the previously obtained matching results. COMA++ [1] is an extended and improved update of the COMA system. It supports ontologies as inputs and provides several matchers for ontology matching.

Compared to other ontology alignment tools, COMA++ produces also very good results in the area of ontology alignment [18]. In principle, ontology matching can use the same ideas as schema matching (e.g. a combination of linguistic and structural matchers). However, as ontologies contain usually more semantic information and constraints than schemas, methods for ontology matching can also use this information to detect similarities between ontologies [11,20]. For example, in [6] a metric for the similarity of concepts is defined using properties, restrictions, sub- and superclass relationships, and so on. There are several tools for ontology alignment, which were also evaluated in the Ontology Alignment Evaluation Initiative 2006 (http://oaei.ontologymatching.org/2006/). Some of them performed better than COMA++ (e.g. Falcon-AO [8] and RiMOM [16]), but these tools are not able to match XML schemas with ontologies.

The ARTEMIS tool [3] for schema integration comes closest to our approach. It also uses a generic metamodel (called reference data model in their work) which is the relational metamodel with some additional object oriented features. Using this generic metamodel, they can uniformly analyze models represented as relational, EER or object oriented models. The matching component matches elements based on their name, data type or structural similarity. To deal with synonyms and hypernyms, the tool uses also an external thesaurus (WordNet). However, ARTEMIS has not been applied to match OWL ontologies and XML schemas. As the tool is not available anymore on the Internet, we could not compare it with our matching system. To the best of our knowledge, COMA++ is the only tool, which is available for download and allows to match XML schemas with OWL ontologies.

3 The Generic Metamodel *GeRoMe*

The *Ge*neric *Ro*le based *Me*tamodel *GeRoMe* [12] uses *role based* modeling. Each model element of a native model (e.g. an XML Schema or a relational schema) is represented as an object that plays a set of roles which decorate it with features and act as interfaces to the model element. We wiil briefly introduce the main ideas of *GeRoMe* by using an example representing an XML schema, which we will use also later as a running example. Representing XML Schema in a generic metamodel is quite challenging as it supports modeling constructs which are not common in other metamodels, such as ordered elements or the derivation of simple types by regular expressions. The example schema contains three complex types (AirlineType, EmpType, and PilotType) and three elements (Airline, Employee, and Pilot). Employees work for an airline, pilots are modeled as a subtype of employee and have an additional attribute Lic_Num.

The *GeRoMe* representation in fig. 1 shows each model element as a *ModelElement* object (gray rectangle) which plays a number of roles (white squares). Each such role object may be connected to other roles or literals, respectively. For the sake of

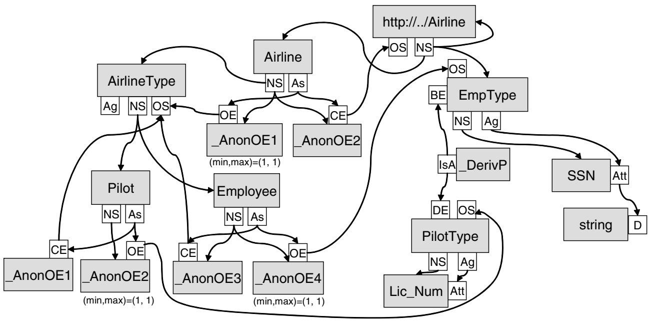


Fig. 1. *GeRoMe* representation of an XML schema

readability, we refrain here from showing the whole model and omitted repeating structures with the same semantics such as *Visible* roles.

The XML Schema element is an association between its enclosing type and the complex type of the nested element. It is always a 1:n association since an XML document is always tree structured. Because of this, the elements in XML Schema are represented by associations in *GeRoMe*. In the example, the elements Airline, Employee and Pilot play *Association* (As) roles connecting the model elements corresponding to the complex types AirlineType, EmpType, and PilotType via anonymous *ObjectAssociationEnd* (OE) and *CompositionEnd* (CE) roles. The *CompositionEnd* role refers to the enclosing complex type of the XML element. The root element Airline is a special case; as it is not enclosed in a complex type, the *CompositionEnd* role for the enclosing type points to the model element http://../Airport representing the schema.

Model elements defined within other model elements such as attributes and XML elements are referenced by the *Namespace* (NS) role of the containing element. For example, the element Employee is owned by the Namespace role of *AirlineType*. Furthermore, the complex types play *Aggregate* (Ag) roles, as they can have attributes, and *ObjectSet* (OS) roles, as they can participate in associations. For example, the *Attribute* (Att) roles of SSN and Lic_Num are connected to the *Aggregate* role of the corresponding model element. Finally, the subtype relationship between PilotType and EmpType is represented by a separate anonymous model element _DerivP. This model element plays an *IsA* role which is connected to the *BaseElement* (BE) role of EmpType and to the *DerivedElement* (DE) role of PilotType.

We have to admit that the *GeRoMe* representation of a model is not easy to understand, but this representation is used only internally in a model management tool; the user will still use her favorite modeling language. The complexity of *GeRoMe* is caused by the complexity of the original modeling languages which can be represented in *GeRoMe*. To be able to represent the details of several modeling languages in a generic way, these details have to be present in *GeRoMe* as well.

The benefit of this generic and detailed representation is that modeling constructs from different metamodels which have equivalent (or similar) semantics are represented by the same roles in *GeRoMe*. This means that the structure of *GeRoMe* models, even if they are originally represented in different metamodels, is similar if they model the same domain. This structural similarity is very important for schema matching as we will show in the evaluation of our matching system in section 5.

4 Schema Matching in *GeRoMeSuite*

Based on the generic metamodel *GeRoMe*, we implemented a schema matching system with the aim to have an extensible and flexible framework for matching models regardless of the modeling language they are represented in. Our system can use any model that can be imported into the generic modeling language as input for the match operation. Currently, this includes relational models, XML Schemas and OWL ontologies.

Another requirement was that the system is built up from components that can be easily combined to new composite algorithms. *GeRoMeSuite* contains a set of *graph traversal* strategies that provide different views on the same model. For each provided graph traversal, there is a corresponding *tree traversal* that can be used if a tree structure is needed. These different views on the structure of the same model influence the results of structure level matchers. Besides allowing variations of the data structure being matched, our matchers also consist of arbitrarily combinable and parametrized steps. In the following, we explain some traversal strategies, the central components of a matcher in *GeRoMeSuite* and how to combine these components to a matcher configuration.

4.1 Graph Traversal Strategies

During the development of the matching system, our aim was to exploit the special characteristics of models represented in *GeRoMe*. As shown in the example in section 3, a *GeRoMe* model is a highly connected structure, i.e. model elements are linked by many different types of relationships. These different types of relationships can be used to define the structure used by the structure level matchers. For example, the links between *Association*, *Aggregate*, and *Attribute* roles could be used to define such a structure. However, there are also other possibilities: the structure implied by the *Namespace* roles define the context in which model elements are defined; *IsA* and other derivation roles can be used to build up a type hierarchy.

In order to use these different structures in our matcher, we defined different iterators for *GeRoMe* models which implement certain traversal strategies, i.e. they navigate a model in a specific way. Our current implementation provides five traversal strategies:

Namespace: Uses the *Namespace* roles to navigate the model.
Derivation: Builds a type (or class) hiearchy.
Association: Uses mainly *Association* roles to navigate the model (e.g. XML Schemas are represented as trees as in most XML editors).
Types: Like the *Association* iterator, but includes also the model elements representing types (e.g. *Aggregates*, *ObjectSets*, *Domains*).
Structure: Reproduces the complete structure of a *GeRoMe* model.

These iterators usually produce graph structures; for matchers which require tree structures as input, we implemented a “meta”-iterator which produces a tree structure from a graph. In addition, we can restrict the iterators to return only model elements which

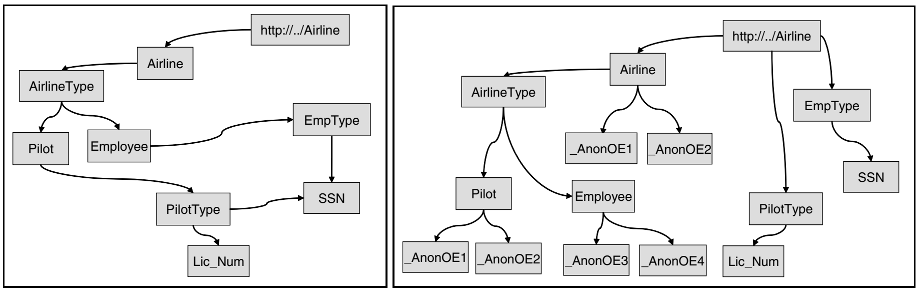


Fig. 2. Types and Namespace traversal for the XML schema from fig. 1

play a *Visible* role (i.e. elements which have an explicit name). Thus, in total we provide twenty different iteration strategies.

Fig. 2 shows the *Types* and *Namespace* traversals for the model from fig. 1. As it can be seen from the example, the traversals imply a different structure as the semantics of the relationships considered in a traversal strategy is quite different. The namespace structure is sometimes an “artificial” structure as it does not represent the structure of the data; it is just the structure in which the schema is defined. The right part of fig. 2 shows the namespace iterator, in which EmpType and PilotType are directly connected to the root of the schema, although the data of these would be nested under an Airline element. A traversal of the model corresponding to the structure of the instance data is produced by the *Types* traversal strategy which is shown in the left part of fig. 2.

4.2 Matcher Components

Schema-based matchers can be classified into element-level and structure-level matchers [23]. Element-level matchers consider an element in isolation, whereas structure-level matchers also consider the context of an element. Additionally to these two kinds of matcher components, *GeRoMeSuite* provides different strategies for aggregation of multiple input morphisms to one output morphism and different filters for morphisms.

Element Level Matchers are capable of computing an initial morphism between two models from scratch. They get two models as input and return a morphism between these two models. Usually, they are based on assessment of similarity for pairs of single model elements. Most such matchers perform a string comparison on the names of the elements using some metric. Whereas this assessment depends in most cases on the two elements alone, it is possible to incorporate the model structure into this step as well. For instance, the similarity of two elements may be determined by the similarity of the possible paths to this element through the model.

GeRoMeSuite provides two basic element level matchers. The `StringMatcher` compares two model elements without taking into account their structure. It is parameterized with a metric that gets two model elements for input and returns a similarity value. Currently, we provide the Levenshtein metric [15] (or edit distance), the Jaro/Winkler metric [10,27], and an improved string matcher [26]. In the future we also plan to add a datatype matcher that can assess the similarity of primitive datatypes.

On the other hand, the `NamepathMatcher` also takes into account the structure of the two models to be matched. It applies the aforementioned string similarity metrics to a set of path expressions that lead to a model element. As its similarity assessment is based on paths to the respective model elements it also requires a tree traversal as a parameter. To assess similarity of two model elements, the `NamepathMatcher` computes the similarity of each pair of paths to the elements and then combines (based on a configurable strategy) these values to the final similarity assessment.

Structure Level Matchers refine an input morphism based on some strategy and on the structures of the models to be matched. That is, they receive a single morphism as input and return a single morphism as output. The general idea of structure level matchers is that the similarity of neighboring elements contributes to the similarity of the element itself. This idea is, for example, realized in the Similarity Flooding algorithm [19] which is also implemented in our system. In addition, our schema matcher provides a *children* matcher. The children matcher resembles the idea of the Cupid algorithm [17]; if the children of an element A are similar to the children of an element B, then A and B are also similar. Both structure level matchers require a graph traversal as input. Furthermore, they are composed of a variety of exchangeable strategy objects that implement certain parts of the respective matching algorithms.

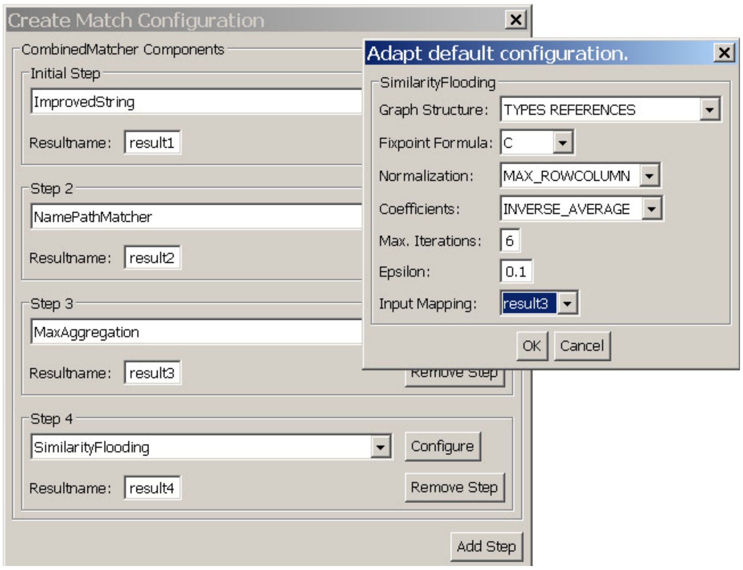
Our schema matching system relies on well-known schema matching methods. The goal of this work is not to provide new algorithms for schema matching, but the usage of a generic metamodel for schema matching and the proof that the generic representation of models is beneficial.

Aggregation Strategies can be used to combine an arbitrary number of morphisms to a single morphism using average, maximum or weighted similarities of model elements.

Morphism filters select similarity values from morphisms based on various criteria such as the maximum distance to the best match, keeping only at least the best K matches, or applying a simple threshold to the similarity values. These filters can be adjusted for existing morphisms to mask or unmask links, but they can also be used as an intermediate step in a matcher to refine the input of subsequent steps.

4.3 Matcher Configuration

Fig. 3 shows an example of how to configure a matcher using the aforementioned components. An arbitrary number of matcher components can be chosen from the set of all matchers already defined by the user and the predefined matchers. In the same way filter and aggregation steps are added to the matcher. Each component has a result morphism and one or more input morphisms. Furthermore, each matching component may provide a GUI class that fills a configuration window with its own controls for specification of its parameters. When all required parameters have been defined the matcher configuration is stored in a configuration repository and is then available for execution and as a component of future custom matchers.


Create Match Configuration
CombinedMatcher Components
Initial Step
Step 2
Step 3
Step 4
Configure
Remove Step
Add Step
Adapt default configuration.
OK
Cancel

Initial Step: ImprovedString
Resultname: result1
Step 2: NamePathMatcher
Resultname: result2
Step 3: MaxAggregation
Resultname: result3
Step 4: SimilarityFlooding
Resultname: result4

SimilarityFlooding
Graph Structure: TYPES REFERENCES
Fixpoint Formula: C
Normalization: MAX_ROWCOLUMN
Coefficients: INVERSE_AVERAGE
Max. Iterations: 6
Epsilon: 0.1
Input Mapping: result3

Fig. 3. Creating a matcher configuration

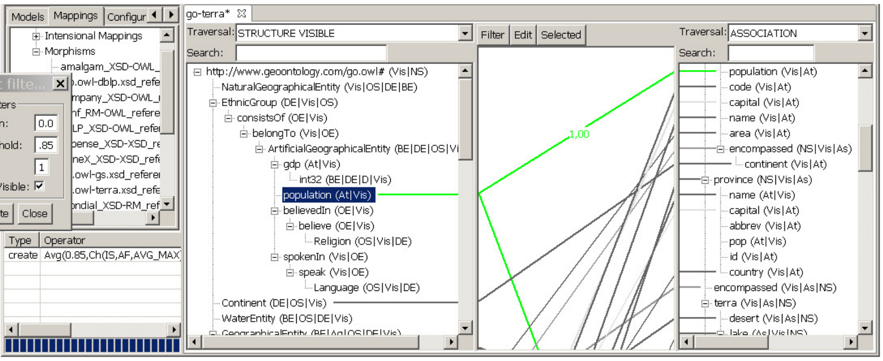
Extensibility. Our matching subsystem is easily extensible. Predefined components such as the different graph and tree traversals or the metrics can be reused for new component classes. All interfaces of the available matcher steps are clearly defined and consolidated. For instance, adding a new filter requires only the implementation of the filter functionality (currently the largest is 32 LOC) and the provision of the user interface (currently the largest is 25 LOC).

Adding new matching algorithms is also easily possible, it just requires the creation of a subclass of an abstract *Matcher* class and the implementation of the *match* method. For example, the implementation of Similarity Flooding uses less than 1000 LOC of which the largest part is used for the implementation of the propagation graph.

For all components, the user interface definition only consists of filling a panel with controls and adding event listeners that update parameter values in the configuration object. This panel is then available in various stages of the process. For instance, a filter can be used for filter steps of a matcher and for filtering a currently displayed morphism.

User Interface. Fig. 4 displays the view of a morphism as it is shown after executing a matcher or loading an existing mapping. Both models are shown as a tree view. The traversal strategy to be used for the tree view can be chosen from a drop-down box.

The morphism itself is shown as a set of lines between the elements of the two models in the center of the view. As in other matching systems different color shades are used to distinguish different degrees of similarity. Links adjacent to selected model elements are displayed in another color. Furthermore, the link(s) with the maximum similarity originating from the selected element is (are) distinguished. To further improve the usability of the system, the user can mask all links that are not adjacent to the currently selected element.



All Filters
Epsilon: 0.0
Threshold: .85
K: 1
Only Visible: ☑

Type	Operator
create	Avg(0.85,Ch(IS,AF,AVG_MAX

Traversal: STRUCTURE VISIBLE
Search:

Traversal: ASSOCIATION
Search:

Fig. 4. Viewing and tweaking a morphism

Using a non-modal filtering dialog, all available filters can be adjusted to filter the currently selected morphism. The filters can be freely narrowed and relaxed until a satisfactory result is found before the user starts to manually fine-tune the morphism.

5 Evaluation

The matcher component has been evaluated by gaging the metrics that are usually used for evaluation of schema matching applications [4], that is *precision*, *recall*, *f-measure(0.5)*, and *overall*. The *overall* metric was developed especially for schema matching systems [4]; it should represent the effort to correct the mapping. As adding mappings is more difficult than removing incorrect mappings, it puts more emphasis on recall than on precision.

For the purpose of this paper, we evaluated only examples that involved ontologies and XML schemas. However, we tested our matching system also with several other examples (also involving other modeling languages) which had a similar results in terms of matching performance as the examples shown in here. As COMA++ is the only other system which is able to match XML schemas and ontologies, and is available for us, we could compare our matching system only to COMA++.

The featured tasks are matching `terra.xsd` from the Mondial data set (`http://www.dbis.informatik.uni-goettingen.de/Mondial/`) with a manually created ontology, matching MapOnto's `DBLP.xsd` with a bibtex ontology (`http://cse.unl.edu/ scotth/SWont/bib.owl`), and the company example (`company.xsd` and `company-er.owl`) from the MapOnto project (`http://www.cs.toronto.edu/semanticweb/maponto/`). For all these tasks and configurations tested, our matching system had an execution time of less than 10 seconds.

5.1 Comparison with COMA++

For COMA++ we performed each of these matching tasks with all available combinations of preconfigured matchers and additionally defined new matchers to search for the

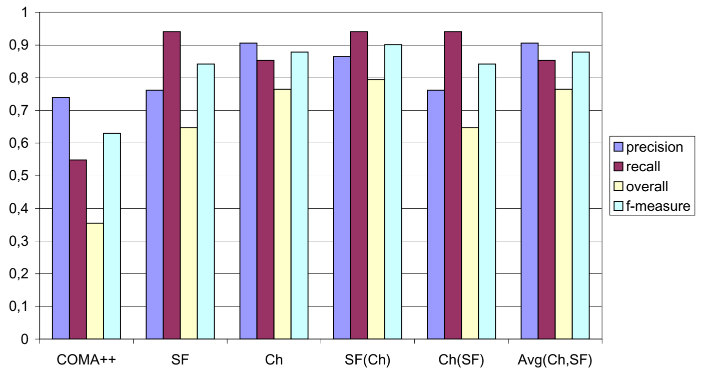


Fig. 5. Comparison of *GeRoMeSuite* with COMA++ for the company example

best possible results. In *GeRoMeSuite* we used basically five different combined matchers. For each of the matchers we used the improved string metric to create an initial match result which was given as input to either the children matcher (Ch) or our similarity flooding implementation (SF). We placed our focus on varying the parameters of these structure level matchers such as traversal strategies or combination of component results to an overall result of the respective matcher. In a next step, we combined these basic matchers in various ways in which we used the best results of the children matcher as input for similarity flooding (SF(Ch)) or vice versa (Ch(SF)) or simply combined the individual result morphisms by computing their average (Avg(Ch, SF)). The following diagrams show the best results of each matcher on the respective match task.

Fig. 5 presents the results of matching the company example, using the metrics precision, recall, overall, and f-measure for COMA++ and the five matchers defined with *GeRoMeSuite*. The company example is a pair of two relatively small models and most elements of the models could be mapped. For COMA++ the best results could be achieved using variations of the original COMA algorithm with different thresholds or other variations of selection strategies. Each of the five matchers of *GeRoMeSuite* outperforms the best result of COMA++ for all quality metrics. Similarity flooding in our implementation achieved better results than the best configuration of COMA++, but was outperformed by the children matcher. However, the best result could be achieved by using the result of the children matcher as initial result for the similarity flooding algorithm (SF(Ch)). The children matcher used the *Association* iteration strategy on this example.

Fig. 6 displays the quality of results for the bibtex/DBLP example. On this example, both tools did not achieve outstanding results. The reason for the poor performance of all matchers is that this matching task is quite difficult as labels and structures of the two models are quite different. *GeRoMeSuite*'s children matcher (Ch) outperformed the best result of COMA++. Whereas its recall is slightly worse, its precision is better by about the same degree. Because the overall metric punishes precision below 0.5, our overall is slightly better. However, the difference is small enough that it seems reasonable to state that both matchers achieve about the same performance. Similarity flooding achieved a very small overall measure due to its low precision on this example.

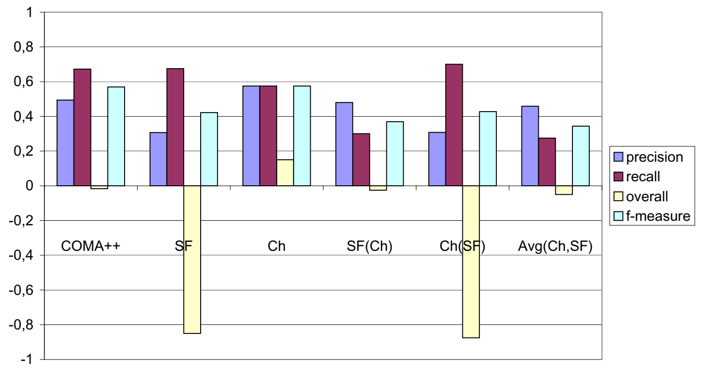


Fig. 6. Comparison of *GeRoMeSuite* with COMA++ for the bibtex example

Consequently, the children matcher that receives similarity flooding's results as input (Ch(SF)) performs poor as well. Overall the simple children matcher returned the best result for this example.

The last example is the task of matching the XML Schema `terra.xsd` from the Mondial database with an ontology of the geographical domain. The results are shown in fig. 7. Again, *GeRoMeSuite*'s children matcher by far outperformed the best result of COMA++. Also, similarity flooding was outperformed by the children matcher. However, the averaging of the two results (Avg(Ch,SF)) slightly dominates both input morphisms. The children matcher used the *Association* traversal strategy, similarity flooding used the *Structure* traversal strategy on this example.

Thus, on the given matching tasks *GeRoMeSuite* was at least as good as COMA++ or even outperformed COMA++. However, we must emphasize that we are of course not as familiar with COMA++ as we are with our own matcher component. There is a large number of configuration options for COMA++ and, consequently, an experienced user may have produced better results with this tool. Nevertheless, we tested more than 50 configurations for COMA++ and presented here only the best results. We tried every

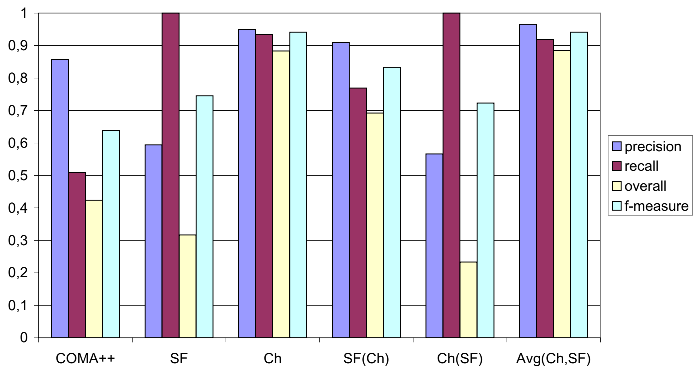


Fig. 7. Comparison of *GeRoMeSuite* with COMA++ for the geographic example

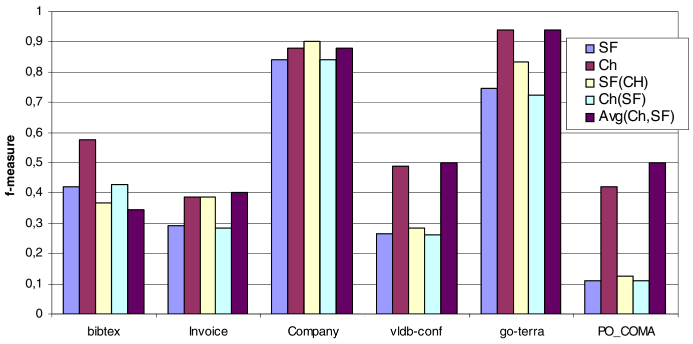


Fig. 8. Quality of matcher results in *GeRoMeSuite* (F-Measure)

configuration using the default matchers and also created some custom matcher configurations searching for more promising results. It is reasonable to assume that comparable results can be achieved for other examples.

In the last example it could be seen that averaging of the two results (Avg(Ch,SF)) dominated both, children matcher and similarity flooding. In fact, our tests on other examples suggest that averaging the results of these two matchers improves the result in many cases.

Fig. 8 compares the results of our matchers in *F-Measure*(0.5) for different matching tasks. The combined matchers SF(Ch) and Ch(SF) could not challenge the children matcher alone. However, the simple aggregation by averaging resulted only in one case (bibtex) in a mapping that was inferior to the input mappings, but in all other cases the results had the same or even better quality than the individual matchers alone.

5.2 Effect of Filter Configuration on the Quality

The variation of morphism filters has of course a significant impact on the quality of the result. *GeRoMeSuite* provides four filters for morphisms. The *epsilon* filter allows all links originating at a model element the confidence of which is within a specified range from the element's best match, the *TopK* filter allows only the best *k* matches for each element, and the *threshold* filter allows links with a confidence measure exceeding a certain value. These filters can be freely configured, whereas the *visible* filter, when enabled, denies all links that involve anonymous model elements such as an anonymous object property that is mapped to a visible property in the other model.

We made the experience that our system is quite stable with respect to variations of the filters, i.e. the results do not vary too much if different filter configurations are applied. Furthermore, the evaluation has shown that if we choose a threshold of about 0.8, the quality of the match result is very close to the best result which can be achieved with our matcher.

For instance, fig. 9 shows the results of adjusting the threshold filter in *GeRoMeSuite* for the company example. The graph shows the results for the children matcher. The best results are those already displayed in fig. 5. We varied the thresholds with steps of 0.05 in an interval from 0.3 to 0.95. The optimal values are reached at a threshold value

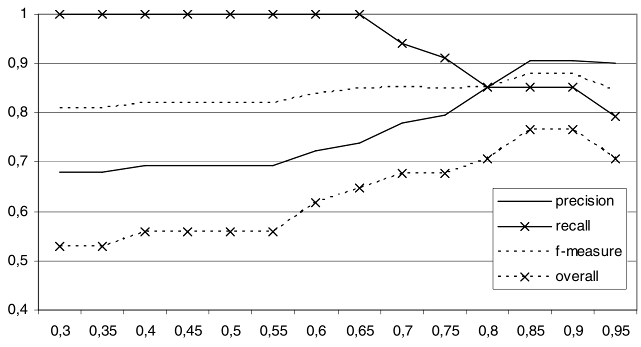


Fig. 9. Matching the company example with different thresholds

of 0.85 and 0.90, respectively. However, the results for a threshold of 0.8 or 0.95 were not considerably worse. For most of the examples we have tested, the best result in terms of f-measure and overall value was produced with threshold values of 0.75 to 0.95.

The stability of the result quality of our matching system with respect to the configuration options is important if “real” matching problems have to be solved, i.e. without having a reference mapping to figure out the best configuration parameters. Using the configuration mentioned above for matching ontologies with XML Schemas, we are confident that the quality of the result is very close to the best result that could be produced with *GeRoMeSuite*.

5.3 Effect of Traversals on the Quality

In section 4, we explained our approach of providing different structural views on the same model. Using traversal strategies, structural matchers can be applied to these different structures, which has an effect on the matching results.

Fig. 10 displays the effect on the quality of results of the same Similarity Flooding matcher which uses different traversal strategies to compute its propagation graph. These are results of matching the geographical example. All matchers had identical configurations except for the traversal strategy. Furthermore, the same filters have been applied to all matcher results. The traversal strategies used were *Association* (A), *Structure* (S) and *Types* (T) and variations of these traversals that omit anonymous model elements from the graph (AV, SV, TV). It can be seen that different graph structures which induce different propagation graphs result in different morphism quality. However, the impact of different traversals on the match result is less than expected. This is probably due to the fact that the similarity of elements in the examples we have tested is mainly determined by the similarity of their labels. Structural similarity has only a small effect on the match result. This sounds like a counter argument to the idea of structural matchers, but the dominance of string matchers is a particular characteristic of the examples we have chosen. We plan further evaluations on this topic in the context of the Ontology Alignment Evaluation Initiative (`http://oaei.ontologymatching.org/2007/`) which also includes test cases in which only the structural similarity can be used.

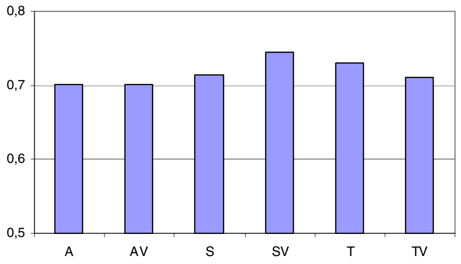


Fig. 10. Performance of Similarity Flooding Using Different Traversal Strategies

5.4 Discussion of the Results

To conclude, for matching ontologies with XML Schemas the children matcher alone or the average of the results of the children matcher and the Similarity Flooding algorithm are a good matcher configuration. The children matcher performed best using the *Association* traversal strategy whereas for similarity flooding the *Structure* traversal strategy was the best choice.

As we implemented only well known schema matching algorithms, the differences in the results of the tools must stem from their internal model representations. For matching ontologies with models from different native modeling languages, the usage of *GeRoMe* as a generic data structure seems to be beneficial. From our experience in the development of the Protoplasm prototype [2], we know that models of different metamodels are represented significantly different when no generic modeling language is used. The graphs represent rather the syntactical structure than the semantics of a model. For example, different labels are used for the edges, and also nodes representing predefined modeling constructs (such as “OWL Class” and “ComplexType”) can have different labels, although the semantics of the model elements is quite similar.

The internal graph structures are not exposed by COMA++, but based on the publications [1,5] and our experience with Protoplasm [2], we can infer that the internal graphs are similar to the graphs shown in fig. 11. The graphs represent the XML schema from section 3 (right part of the figure), and an ontology for the same domain.

These graphs might be easier to understand for humans than the *GeRoMe* representation in section 3. However, as we are dealing with *automatic* schema matching methods, human-readability is not an issue. For a schema matching tool, the graphs contain some problems. First of all, the labels of the edges are different except for the “type” edge. Identical edge labels are for example an important requirement for the Similarity Flooding algorithm as its main data structure, the propagation graph, is build according to identical edge labels in the two graphs. If the labels are different, then the propagation of similarity values to neighboring nodes does not work.

Furthermore, the structure of the graphs is different although the same domain is represented. For example, the association between Airline and Employee/Pilot is not directly visible in the XML schema. Thus, the structural similarity will be considered as very low.

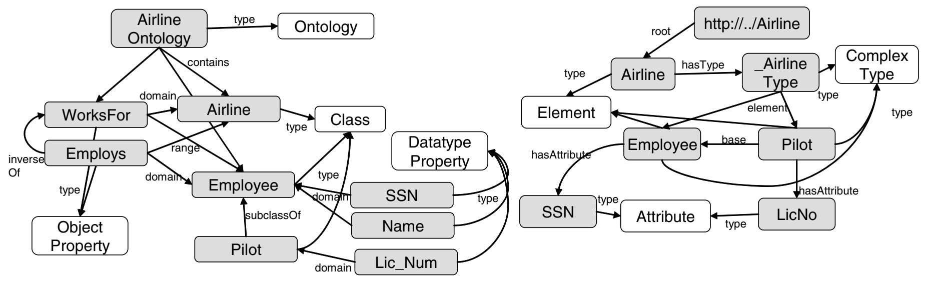


Fig. 11. Graph representation of an ontology and an XML schema

6 Conclusion

We implemented a schema matching subsystem for our holistic model management system *GeRoMeSuite* [13] to match models represented in different metamodels. Our results show the usefulness of our generic metamodel *GeRoMe* for generic model management tasks. The matcher returned comparatively good results when matching models represented in different modeling languages. The comparison with COMA++, another matching system capable of matching XML schemas and OWL ontologies, has shown that *GeRoMeSuite* achieved better results in all test cases. Our system provides several algorithms for element level and structure level matchers; these basic matchers can be combined in a very flexible way which enables the definition of arbitrary matcher combinations. The evaluation has shown that the combination of matchers leads often to better results than the individual matchers.

Furthermore, our matching system is quite stable with respect to different scenarios and configuration options. Using a reasonable combination of matchers and a high threshold value produces a result which is close to the best result that can be achieved with our matcher. Thus, the application of our system to new scenarios can use a standard configuration. Therefore, the user does not need to have a deep understanding of the system, and can still expect a good result of the matching system.

Our results suggest that the usage of a generic metamodel can improve the performance even of model management operators that do not rely on detailed semantics of metamodel constructs, such as the Match operator. Algorithms for matching models are usually interested only in properties of individual nodes, such as labels or types, and in the abstract graph structure of the model. However, the unification of structure that comes along with using a generic metamodel improves their results. Our matching system provides also various traversal strategies for models, and is not restricted to one graph representation of the model. Depending on the structural information available, the user can choose an appropriate traversal strategy (e.g. IsA hierarchy, associations).

In the near future we plan to improve the usability of our matcher application. Improving the usability and visualization in matching systems is itself an active research area [24]. Automated focussing of matching elements, collapsing and expanding trees when exploring a mapping are already included in our current prototype. We also plan to provide algorithms for sorting the children of tree nodes such that the number of line crossings is minimized. This would highly increase the readability of morphisms.

As our matching subsystem is very easily extendable, it forms a thorough basis for further research on schema and ontology matching. Therefore, we will also implement and evaluate more and new matcher components, and apply them to other homogeneous and heterogeneous matching scenarios. The currently implemented matching components are general purpose components that we can apply to any kind of models. Our next steps include the definition of special purpose matcher components that exploit the characteristics of particular metamodels, e.g. OWL ontologies.

Acknowledgements. This work is supported by the DFG Research Cluster on Ultra High-Speed Mobile Information and Communication (UMIC, http://www.umic.rwth-aachen.de).

References

1. Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and ontology matching with COMA++. In: Proc. SIGMOD Conf., pp. 906–908. ACM Press, New York (2005)
2. Bernstein, P.A., Melnik, S., Petropoulos, M., Quix, C.: Industrial-Strength Schema Matching. SIGMOD Record 33(4), 38–43 (2004)
3. Castano, S., Antonellis, V.D., di Vimercati, S.D.C.: Global Viewing of Heterogeneous Data Sources. IEEE Transactions on Knowledge and Data Engineering 13(2), 277–297 (2001)
4. Do, H.H., Melnik, S., Rahm, E.: Comparison of Schema Matching Evaluations. In: Aksit, M., Mezini, M., Unland, R. (eds.) NODe 2002. LNCS, vol. 2591, pp. 221–237. Springer, Heidelberg (2003)
5. Do, H.H., Rahm, E.: COMA - A System for Flexible Combination of Schema Matching Approaches. In: VLDB. Proc. 28th Intl. Conf. Very Large Data Bases, pp. 610–621 (2002)
6. Euzenat, J., Valtchev, P.: Similarity-based ontology alignment in OWL-Lite. In: ECAI 2004. Proc. 16th European Conference on Artificial Intelligence, pp. 333–337 (2004)
7. Hernández, M.A., Miller, R.J., Haas, L.M.: Clio: A Semi-Automatic Tool For Schema Mapping. In: Proc. ACM SIGMOD, p. 607 (2001)
8. Hu, W., Cheng, G., Zheng, D., Zhong, X., Qu, Y.: The Results of Falcon-AO in the OAEI 2006 Campaign. In: Intl. Workshop on Ontology Matching (OM-2006), Athens, GA, USA (2006)
9. ISO/IEC. Information technology – Information Resource Dictionary System (IRDS) Framework. International Standard ISO/IEC 10027, 1990 (1990)
10. Jaro, M.: Probabilistic linkage of large public health data files. Statistics in Medicine 14, 491–498 (1995)
11. Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: the state of the art. The Knowledge Engineering Review 18(1), 1–31 (2003)
12. Kensche, D., Quix, C., Chatti, M.A., Jarke, M.: *GeRoMe*: A Generic Role Based Metamodel for Model Management. Journal on Data Semantics VIII, 82–117 (2007)
13. Kensche, D., Quix, C., Li, X., Li, Y.: *GeRoMeSuite*: A System for Holistic Generic Model Management. In: Proc. 33rd Int. Conf. on Very Large Data Bases (to appear, 2007)
14. Kensche, D., Quix, C., Li, Y., Jarke, M.: Generic Schema Mappings. In: ER 2007. Proc. 26th Intl. Conf. on Conceptual Modeling (to appear, 2007)
15. Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10, 707–710 (1966)

16. Li, Y., Li, J., Zhang, D., Tang, J.: Result of Ontology Alignment with RiMOM at OAEI 2006. In: Intl. Workshop on Ontology Matching (OM-2006), Athens, GA, USA (2006)
17. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic Schema Matching with Cupid. In: Proc. 27th Intl. Conf. on Very Large Data Bases (VLDB), Rome, Italy, pp. 49–58 (2001)
18. Massmann, S., Engmann, D., Rahm, E.: COMA++: Results for the Ontology Alignment Contest OAEI 2006. In: Intl. Workshop on Ontology Matching (OM-2006), Athens, GA, USA (2006)
19. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In: Proc. 18th Intl. Conference on Data Engineering (ICDE), pp. 117–128. IEEE, Los Alamitos (2002)
20. Noy, N.F.: Semantic Integration: A Survey Of Ontology-Based Approaches. SIGMOD Record 33(4), 65–70 (2004)
21. Object Management Group. Common Warehouse Metamodel (CWM), version 1.0. Spezifikation (February 2001)
22. Quix, C., Kensche, D., Li, X.: Generic Schema Merging. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007. LNCS, vol. 4495, pp. 127–141. Springer, Heidelberg (2007)
23. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal 10(4), 334–350 (2001)
24. Robertson, G.G., Czerwinski, M., Churchill, J.E.: Visualization of mappings between schemas. In: CHI. Proc. Conf. on Human Factors in Computing Systems, pp. 431–439. ACM, New York (2005)
25. Shvaiko, P.: A Survey of Schema-Based Matching Approaches. Journal on Data Semantics IV IV, 146–171 (2005)
26. Stoilos, G., Stamou, G.B., Kollias, S.D.: A String Metric for Ontology Alignment. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 624–637. Springer, Heidelberg (2005)
27. Winkler, W.: The state record linkage and current research problems. Tech. rep., Statistics of Income Division, Internal Revenue Service Publication (1999)

Labeling Data Extracted from the Web

Altigran S. da Silva[1], Denilson Barbosa[2],
João M.B. Cavalcanti[1], and Marco A.S. Sevalho[1]

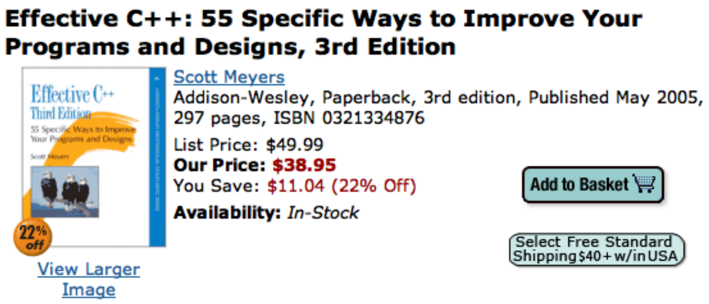
[1] Universidade Federal do Amazonas
Manaus, AM, Brazil
{alti,john,msevalho}@dcc.ufam.edu.br
[2] University of Calgary
Calgary, AB, Canada
denilson@ucalgary.ca

Abstract. We consider finding descriptive labels for anonymous, structured datasets, such as those produced by state-of-the-art Web wrappers. We give a probabilistic model to estimate the *affinity* between attributes and labels, and describe a method that uses a Web search engine to populate the model. We discuss a method for finding good candidate labels for unlabeled datasets. Ours is the first unsupervised labeling method that does not rely on mining the HTML pages containing the data. Experimental results with data from 8 different domains show that our methods achieve high accuracy even with very few search engine accesses.

1 Introduction

The Web is a vast, albeit disorganized, source of valuable information. To extract such information into a format suitable for use by other applications, several Web wrappers have been proposed. However, these methods [1,3,16] recognize only the structure, but not the *semantics*, of the Web data: They produce *anonymous* datasets (i.e., datasets with meaningless labels in their schema). This is unfortunate, as data integration tools often rely on the existence of meaningful labels in the schema [11]. In face of these limitations, other authors have proposed methods for *labeling* anonymous data extracted by Web wrappers [2,4,13]. In general, these methods work by mining terms with distinctive formatting within the original pages containing the data. While high accuracy is sometimes achieved (the authors of [2] report up to 90% accuracy), this approach has two drawbacks. First, typical Web pages often omit labels, which are understood from the context (by a human). For instance, the book description in Figure 1 contains some labels (e.g., ISBN), while others are missing (e.g, title and publisher). Second, and more importantly, this approach restricts one to using only those labels chosen by the Web content providers, which may not be the most appropriate or most descriptive ones.

We propose a novel and highly effective method for automatically labeling anonymous data based on a simple probabilistic model that takes into account the affinity between a set of values (i.e., an anonymous attribute) and potential attribute labels. The probabilities are estimated by counting the number of answers to *speculative queries*, obtained from a standard Web search engine. Intuitively, a speculative query formulates a hypothesis that a given term is a good


Effective C++: 55 Specific Ways to Improve Your Programs and Designs, 3rd Edition

Scott Meyers
Addison-Wesley, Paperback, 3rd edition, Published May 2005, 297 pages, ISBN 0321334876
List Price: $49.99
Our Price: $38.95
You Save: $11.04 (22% Off)
Availability: *In-Stock*

Fig. 1. Book description extracted from `http://www.bookpool.com`

label for an attribute in the anonymous dataset. The search engine is used as an oracle to determine how *plausible* such a hypothesis is. Unlike previous methods, our method is oblivious to where the candidate labels come from. Also, we give a fully automatic method for finding good candidate labels for an anonymous dataset. Our approach is to search the Web for documents containing certain text patterns commonly used to enumerate instances of classes of objects [6]. We exploit these patters to mine frequently occurring terms that can be used as labels. Finally, we report an experimental evaluation using real Web data from different domains thats shows that our approach is very effective, even when very few accesses to a search engine are employed.

The remainder of the paper is organized as follows. Sections 2 discusses related work. Section 3 introduces the terminology and explains our method. Sections 4 and 5 detail the implementation of our candidate label selection and labeling algorithms, respectively. Sections 6 presents and evaluates our experiments. Finally, we conclude in Section 7.

2 Related Work

There is already a vast literature on automatic and semi-automatic Web data extraction methods. Laender et al. [7] give a recent survey of the topic. While differing in many aspects, all those methods identify only the structure of the Web data, producing *anonymous* datasets, as discussed above.

To the best of our knowledge, there are only two *generic* methods for the automatic labeling of anonymous data: *DeLa* (Data Extraction and Label Assignment) [13] and *Labeller* [2]. (We note in passing that there are accurate *domain-specific* methods for extracting labeled data; e.g., Reis et al. [4] describe an automatic tool for extracting and labeling news articles.) *DeLa* is an automatic data extraction technique for Web pages accessible through seach forms (which form the so-called *deep Web*). It tries to discover a schema for the data and associates labels to the anonymous attributes based on the following: (1) the similarity between labels in the fields of the form with data values in a page accessed through the form; (2) the presence of labels at the headings of HTML tables containing those data values; (3) the presence of labels in the vicinity

R	A_1	A_2
	Miles Davis	Kind of Blue
	John Coltrane	A Love Supreme
	John Coltrane	My Favorite Things
	...	

Fig. 2. An anonymous dataset about music, containing a single relation $R(A_1, A_2)$

of those data values; (4) the formatting of data values, e.g., the presence of an "@" inside a value may suggest "e-mail" as a suitable label for it. Their authors report experimental results showing that the combination of all four heuristics led to almost 90% accuracy in the label assignment.

Labeller extends RoadRunner [3] by annotating the automatically generated wrappers with labels mined from the pages containing the data. This is done with heuristics for finding the best associations between labels and data values based on the graphical rendering of the pages by a browser. These heuristics capture common page design styles, and consider both the distance and the alignment between a data value and its corresponding label. For instance, *Labeller* relies on the fact that a label is usually placed above or to the left of the corresponding data value, in a location never too far from it, etc. Their authors report accuracy results matching those of *DeLa*.

A severe limitation of both previous methods is the dependence on finding the labels within the Web documents from which the data was extracted, which, as discussed above, fails if any of the labels are missing in the Web document (e.g, see Figure 1). Moreover, the approach limits the choice of labels to only those used by the content author, which may not be best suited for the user interested in the data. Our method, on the other hand, does not suffer from either problem. Our probabilistic model estimates the appropriateness of a label regardless of where it come from, allowing the user to provide her own set of labels. Also, we find candidate labels by searching the Web, resulting in commonly used labels. Finally, we show good experimental results with both manually and automatically selected labels (see Section 6).

Interestingly, our probabilistic model for estimating the affinity between attributes and labels is similar to the PMI-IR algorithm proposed by Turney [12], for finding synonyms amongst English words (by estimating their similarity). As observed in that paper, the accuracy of the similarity results depends on the size of the document collection in the underlying Information Retrieval system (potentially the entire Web in our case). Overall, our accuracy results, obtained through experiments on several domains, are very similar to those of Turney.

3 Probabilistic Labeling of Anonymous Data

An anonymous dataset is a structured collection of data in which descriptive labels for similar objects are missing; e.g., Figure 2 shows a (relational) anonymous dataset about music. This dataset is a prototypical example of the output

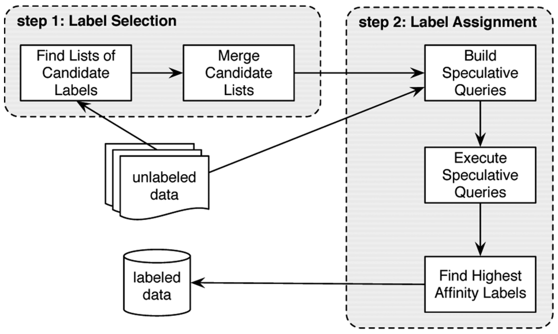

step 1: Label Selection
Find Lists of Candidate Labels
Merge Candidate Lists
step 2: Label Assignment
Build Speculative Queries
Execute Speculative Queries
Find Highest Affinity Labels
unlabeled data
labeled data

Fig. 3. Overview of the proposed approach

produced by most state-of-the-art data extraction tools[1]. Note that the data has a well defined, albeit semantically poor, schema $R(A_1, A_2)$; also, both attributes have well defined domains (A_1 contains artist names and A_2 contains album names). The labeling problem, which we deal with in this paper, consists of finding descriptive labels for an anonymous relational dataset.

As illustrated by Figure 3, the labeling problem can be solved in two steps: Finding a good set of candidate labels, and finding the best matching between labels and the attributes. It is important to separate these steps because different ways of finding candidate labels are possible. For instance, as discussed above, the labels may be mined from the pages with the data or the user may provide her own labels. We propose fully automated *and* independent methods for the label selection and the label assignment tasks in this paper.

3.1 Candidate Label Selection

It has been observed that one can discover instances of a given *class* of similar objects, with high accuracy, by formulating queries to Web search engines [5,15]. The process consists of searching for documents with specific patterns, usually called *Hearst patterns*, expressing relationships among terms, particularly *hyponymy*. For instance, the pattern "NP_1 such as $(\mathsf{NP}_2)*$" can be used to find one or more instances NP_2 (for noun phrase) of a class NP_1. To find city names using this method, one searches for documents containing the phrase "cities such as", and extracts frequently occurring terms that appear after that phrase in those documents. This approach works well in practice when used on sufficiently large text corpora [6], including the Web.

The problem of finding labels for a set of attributes is the converse of the problem above, and can be partly addressed using a similar approach. Before we describe our method, however, it is important to discuss why the use of Hearst patterns alone fails to find good labels (and hence, why step 2 in our method is still necessary). Hearst patterns are designed for finding hyponyms [6]; that is, finding terms that are special kinds of another, more general, term. Thus, one

[1] Some tools produce nested relations encoded as XML, which can be easily converted into a relational format.

is likely to find that Paris is a city using this method, because it is a popular hyponym of city. However, the word Paris refers to many different objects in different contexts. (In fact, at the time of writing, in none of the first 5 documents returned by the Google, Yahoo! and MSN search engines to the query “such as Paris”, the term Paris is used in direct reference to the city. The closest match, occurring in 2 of the 15 documents, refers to Paris as a location, which is a *hypernym* of city.) Moreover, it is not clear how accurate a method based on Hearst patterns would be if the set of values used for finding the associations does not contain very popular terms.

Finding Candidate Labels. Our label selection algorithm (Section 4) is based on the following assumptions. Given attribute A_i of anonymous relation R, we assume that: (1) plausible labels for A_i are likely to occur in Web documents that contain instances of A_i; (2) such labels are likely to appear close to those instances in such documents; (3) this close co-occurrence expresses the hyponymy relationship between the label and the instance; (4) this situation is frequent on the web, so that these pages are likely to be crawled by popular search engines.

We consider three strategies when searching for candidate labels. In all of them, we formulate a query to a search engine using a Hearst pattern and search for nouns that appear close to the exact query phrase in the documents returned by the search engine. The strategies differ only w.r.t. where the search for candidate labels is done, relative to the occurrence of the query phrase. In a *forward* search, we search for candidate labels *after* the query expression. Conversely, in a *backward* search, we look for candidate labels *before* the query text. Finally, in a *bidirectional* search, we search in both directions.

In order to find a good list of candidate labels for the entire dataset we obtain individual list for each attribute separately, which are merged into a single one at the end. We do so because in our experiments, it was not uncommon to find a good label for a given attribute appearing in several individual lists. Regardless of the search strategy used, we assign a numeric score to each candidate label that is inversely proportional to the square of the distance between the label and the query phrase (measured in number of words). Intuitively, this strategy is strongly biased towards labels that are both frequent on the Web *and* often appear close to the given query pattern.

3.2 Affinity-Based Labeling

Let $R(A_1, A_2, \ldots, A_n)$ be a relation on n anonymous attributes $A_1, \ldots, A_n$, where each A_j is of domain D_j, and domains are pairwise disjoint. Assume we are given an instance of R with t tuples, and a set $L = \{l_1, \ldots, l_m\}$ with m *candidate labels* ($m > n$). Our goal is to assign to each A_j a label $l_i \in L$ which is the *best* descriptor for attribute A_j.

There are two challenges in finding good labels for anonymous data. First, we need a way of measuring how well a labeling $R \rightarrow L^n$ describes the (domains of the) attributes in R. Second, the cost of the labeling algorithm must not be too high. (We note in passing that the labeling problem is equivalent to finding a maximum-weight matching in a complete bipartite graph where the vertex sets are the columns in R and L, respectively, and the weight of each edge (A_j, l_i)

indicates how well l_i describes A_j. The fastest algorithms for this problem run in low polynomial, but super-linear, time on the size of the graph [14].)

In order to minimize the number of accesses to the search engine, which are orders of magnitude more expensive than any other operation in our method, we use a greedy labeling strategy: We find a label for each attribute in isolation, and once a label is assigned to an attribute it is no longer considered as a candidate label for other attributes. Also, we take a probabilistic approach for estimating the goodness of a labeling. More precisely, we use $P(l_i \mid A_j)$, the probability of l_i describing well attribute A_j as the metric for evaluating a labeling of an anonymous dataset. By doing so, we account for the inherent uncertainty in how well a label describes a domain. For simplicity, we assume that the probability of a label being a good fit for an attribute is independent of other attributes.

Computing Label-Attribute Affinities. Note that $P(l_i \mid A_j) = \frac{P(A_j|l_i)P(l_i)}{P(A_j)}$. Thus, we need to estimate $P(A_j \mid l_i)$ and $P(l_i)$ in order to compute the affinity between l_i and A_j. ($P(A_j)$ is a normalizing factor, and can be ignored for all practical purposes.) Intuitively, the prior probability $P(l_i)$ captures the user' preference for the label l_i *regardless* of its affinity with any of the attributes.

W approximate the true affinity between labels and domains by submitting *speculative queries* to a standard Web search engine. A speculative query (to be defined later) is a statement saying that a given label l_i is a good descriptor for attribute A_j. We use the *number of documents* that the search engine classifies as relevant answers for that query to estimate the probabilities above. The intuition behind speculative queries is as follows. If label l_i is a better match for attribute A_j than label l_k, a Web document D containing high quality information about an instance of A_j is more likely to refer to l_i than to l_k. A complementary interpretation of our model is to assume that every Web document about a given value from attribute A_j (e.g., a specific artist if the domain of A_j is artist names) is an "expert" on (the entity represented by) that value. Thus, the number of documents in the answer to a speculative query represents the number of experts that consider the given label a good match for the given attribute value.

More precisely, we define:

Definition 1. *The* Document Count *of a query expression e, denoted $DC(e)$, is the number of documents relevant to e according to a given Web search engine.*

Definition 2. *Given an anonymous relation $R(A_1, \ldots, A_n)$ and a set of candidate labels $L = \{l_1, \ldots, l_m\}$, the Label-Attribute Affinity between A_j and l_i, denoted $LAA(A_j, l_i)$, is defined as*

$$LAA(A_j, l_i) = P(A_j \mid l_i) = \left(\frac{1}{\| [A_j] \|}\right) \sum_{x=1}^{\|[A_j]\|} \frac{DC(l_i \wedge v_x)}{\sum_{y=1}^{m} DC(l_y \wedge v_x)}$$

where $[A_j]$ is the active domain[2] *of A_j, and $v_x \in [A_j]$.*

We write $l_i \wedge v_x$ do denote the speculative query asserting that label l_i and value $v_x \in A_j$ are likely to appear together in a Web document. Note that we restrict

[2] Recall the active domain of an attribute is the set of distinct values for the attribute that are used in the actual databases instance.

Table 1. Query patterns used in the Label Selection method

p	region(s_i,p)	query(p,v)
bwd	terms before query expression	"such as v"
$perm$	terms before query expression	"v_1, v_2"
fwd	terms after query expression	"v is "
val	all terms in answer	"v"

the definition of LAA to the *active domain* of each attribute; that is, we ignore duplicate values. Intuitively, this avoids skewing the affinity of an attribute based on the relative frequency of the values in its active domain. For the purposes of this work, we define $P(l_i)$, the *a priori* probability of l_i being a good label for any attribute as the relative frequency of l_i among the candidate label lists obtained in step 1 (recall Section 3.2).

Sampling. Even with a fast Internet connection, invoking a search engine several times to compute $DC(e)$ is a costly operation in terms of clock time. Thus, instead of using $t{\cdot}m$ speculative queries for computing $P(A_j \mid l_i)$ as in Definition 2, we will consider only a sample of the tuples in the anonymous database.

4 Selecting Candidate Labels

We now detail our algorithm for finding candidate labels. We consider only nouns as candidate labels; moreover, we use the Java WordNet Library[3] to recognize variants of the same noun obtained by applying the usual inflection rules. That is, candidate labels in our approach are nouns in the singular, represented in their canonical form according to WordNet. As discussed in Section 3.1, we search for labels in Web documents containing certain text patterns instantiated with instances of the anonymous attribute. We restrict our search space to the document *snippets*[4] returned by the search engine. Intuitively, this restricts the set of candidate labels to only those that appear close to the attribute value in the document.

Table 1 shows the patterns we use. Both the *bwd* (backward) and *perm* (permutation) patterns are used with backward searches (recall Section 3.1), while the fwd (forward) pattern is used with a forward search, and *val* (value) pattern is used with a bidirectional search. Two attribute instances are used with the permutation pattern; our experience indicates that doing so is very effective when dealing with categorical attributes.

Ranking. Let S be the set of snippets of the m highest ranked documents for a query formulated using pattern p with some value v of an anonymous attribute A_i. For a given term t in S, the *coincidence factor* of t and v in S is defined as:

$$\alpha(t, v, S) = \sum_{s_i \in S_t} \frac{W_i^t}{d(t, v, s_i)^2} \quad , \qquad (1)$$

[3] http://jwordnet.sourceforge.net.

[4] Snippets are small text fragments of Web documents returned with the answers of a search engine that usually contain the terms used in the query.

```
1  Input: anonymous relation $R(A_1, \ldots, A_n)$
2  Output: set $L$ of candidate labels for $R$
3  begin
4     foreach $A_i \in \{A_1, \ldots, A_n\}$ do
5        $T_i \leftarrow \emptyset$; $V \leftarrow$ sample with $k$ distinct instances of $A_i$
6        foreach $v \in V$ do
7           $S \leftarrow \emptyset$
8           foreach pattern $p \in \{fwd, bwd, val, perm\}$ do
9              $S \leftarrow S \cup$ top $m$ snippets for query($p$,$v$)
10          end
11          foreach $t$ in a snippet in $S$
12             $\alpha_{curr} \leftarrow \alpha(t, v, S)$
13             if t$\notin T_i$ then
14                $T_i \leftarrow T_i \cup \{t\}$; $\alpha_{max}[t, v] \leftarrow 0$
15             end
16             $\alpha_{max}[t, v] \leftarrow \max\{\alpha_{max}[t, v];\ \alpha_{curr}\}$
17          end
18          foreach $t \in T_i$
19             $score[A_i, t] \leftarrow score[A_i, t] + \alpha_{max}[t, v]$
20          end
21       end
22       $L_i \leftarrow$ $\ell$ terms $t$ in $T_i$ with highest $score[A_i, t]$
23    end
24    $L \leftarrow L_1 \cup \ldots \cup L_n$
25 end
```

Fig. 4. The *Label Selection* Algorithm

where S_t is the subset of snippets $s_i \in S$ that contain the term t in region(s_i,p) (see Table 1), W_i^t is the frequency of term t in the snippet s_i and $d(t, v, s_i)$ is the distance between t and v in s_i.

Intuitively, Equation 1 captures the desiderata for a good label, as discussed in Section 3.1: It favors terms that appear in many snippets, penalizing those that occur "far" from the pattern in the snippet.

4.1 Label Selection Algorithm

Our Label Selection algorithm (Figure 4) works as follows. For each attribute A_i in anonymous relation R, we build a list of candidate labels T_i, keeping in the final answer only those ℓ with highest coincidence factor (line 22). T_i contains all nouns (normalized according to WordNet, as discussed above) that appear in a (snippet of a) Web document close to an instance of A_i.

In order to build T_i, we pick k instances of A_i at random and, for each instance v, collect the document snippets from a search engine using the four search patterns in Table 1 (lines 8–10). Next, we find the *highest* coincidence factor between each term t appearing in any snippet obtained with v (lines 11–17). Finally, we define the final score of term t relative to the entire attribute A_i as the sum of the highest coincidence factors of t relative to *all* sample instances of A_i (lines 18–20).

Table 2. Number of answers to different kinds of speculative queries

label	value	*phrase*	$l + v$	*words*
artist	Miles Davis	39,900	5,980,000	12,000,000
title	Miles Davis	220	1,940,000	16,300,000
album	Miles Davis	9,230	6,140,000	9,170,000
artist	John Coltrane	21,700	2,910,000	3,710,000
title	John Coltrane	83	1,670,000	2,720,000
album	John Coltrane	493	3,340,000	4,180,000
artist	Kind of Blue	7	425,000	25,600,000
title	Kind of Blue	117	539,000	46,700,000
album	Kind of Blue	17,900	885,000	17,900,000
artist	A Love Supreme	36	213,000	9,000,000
title	A Love Supreme	60	158,000	16,800,000
album	A Love Supreme	497	318,000	5,610,000

Analysis. The algorithm in Figure 4 is straightforward. It runs in $O(n \cdot m \cdot k)$ time, where k is the number of samples from each anonymous attribute, m is the number of snippets retrieved for each query, and n is the number of attributes in the dataset. There is a trade-off when choosing k: it must be high enough to ensure that a representative sample of the attribute is used, but not too high, to avoid an excessive number of search engine calls. Conversely, because the snippets returned by the search engine are ranked according to the relevance of the corresponding documents, lower values of m should yield better labels; however, using very low values of m require more queries to be issued in order to obtain a representative set of candidate labels. In preliminary experiments, we obtained better results when both k and m were set to 10; we used this setting in all experiments discussed in the paper.

The last parameter, ℓ, determines the number of candidate labels that will be selected for each anonymous attribute. Obviously, the value of ℓ has an impact not only the accuracy of the labeling method as a whole, but also the cost of the label assignment step (recall Figure 3). Initial experiments we have carried out indicate the $\ell = 10$ is a good choice for this parameter.

A final remark on the implementation of our Label Selection algorithm concerns the elimination of stopwords, punctuation as well as other special characters. Our experiments indicate that doing so has little effect on the accuracy of the method, but substantially reduces the memory footprint of our algorithm, as fewer terms are kept in memory.

5 Speculative Labeling

This section describes our implementation of the labeling method and our strategy to formulate speculative queries. We illustrate the discussion using the dataset in Figure 2 and candidate labels $L = \{\text{artist, title, album}\}$.

Table 3. Label-Attribute Affinities

	phrase	$l + v$	*words*
$P(A_1 \mid \text{artist})$	**0.8911**	0.3964	0.3350
$P(A_1 \mid \text{title})$	0.0043	0.1744	0.3457
$P(A_1 \mid \text{album})$	0.1046	0.4292	0.3193
$P(A_2 \mid \text{artist})$	0.0305	0.2695	0.3160
$P(A_2 \mid \text{title})$	0.0538	0.2604	0.5826
$P(A_2 \mid \text{album})$	**0.9156**	0.4701	0.1014

5.1 Formulating Speculative Queries

Recall that speculative query $l_i \wedge v_x$ formulates the hypothesis that l_i is a class of objects of which v_x is a member. The first question that arises is how to formulate such a hypothesis using the keyword-based search paradigm currently in use in all major Web search engines.

Table 2 compares the number of results of three kinds of speculative queries; all queries were run using the Google search engine on June 3, 2006. Column 3 shows the DC values for the *phrase* approach, which uses speculative queries asking for documents containing a phrase formed by the label and the value (e.g., "artist John Coltrane"). Column 4 shows the DC values for the $l + v$ approach, which uses speculative queries asking for documents containing the label *and* the value (e.g., "artist" and "John Coltrane"). Finally, column 5 shows the DC values for the *words* approach, whose speculative queries ask for documents containing the label and every word in the value (e.g., the words "artist", "John" and "Coltrane").

Table 3 shows the affinity between the labels and attributes, computed as in Definition 2, using the DC values in Table 2. Note that the *phrase* approach yields a very good labeling after considering only two tuples in the anonymous dataset. The $l+v$ approach finds one correct label assignment (that A_2 contains albums), albeit with much lower confidence. Finally, the *words* approach gives a less precise labeling for A_2. One reason why the *words* performs poorly is that speculative queries built in that way tend to be too general; thus, the very high DC values for them. We observed experimentally that $l+v$ and *phrase* yield high DC values for *correct* hypothesis. However, $l + v$ does not discriminate *incorrect* hypothesis well because high quality Web pages on a given domain tend to contain several good labels (e.g., a document with Miles Davis's discography is likely to have all labels in our example). Hence, we used the *phrase* approach only in the experiments reported in this paper.

5.2 The Speculative Labeling Algorithm

Figure 5 shows our *Speculative Query Labeler* algorithm, which works as follows. We iterate over all attributes in R, assigning a label to each of them, one at a time and independently of others. For each attribute A_i, we pick k distinct instances of it (line 7), and, for each instance v and label l_j still available, we compute the $DC(l_j \wedge v)$ (lines 8-13). $LVC[i, j]$ accumulates the document counts of all speculative queries using label l_j and a v value from attribute A_i. The affinity between A_i and each label l_j is estimated, and the best label for A_i (called l_b) is chosen (lines

```
1  Inputs: anonymous relation $R(A_1, \ldots, A_n)$
2          set of candidate labels $L = \{l_1, \ldots, l_m\}$
3  Output: labeling of $R$
4  begin
5    foreach $A_i \in \{A_1, \ldots, A_n\}$ do
6      $b \leftarrow 1;\ sum \leftarrow 0$
7      $V \leftarrow$ sample with $k$ distinct instances of $A_i$;
8      foreach $v \in V$ do
9        foreach $l_j \in L$ do
10         $s \leftarrow$ speculative query $l_j \wedge v$
11         $LVC[A_i, l_j]$ += $DC(s)$; $sum$ += $LVC[A_i, l_j]$
12       end
13     end
14     foreach $l_j \in L$ do
15       $LAA[A_i, l_j] = LVC[A_i, l_j]/sum$
16       if $(P(l_j) \cdot LAA[A_i, l_j] > P(l_b) \cdot LAA[A_i, l_b])$
17         $b \leftarrow j$
19       end
20     end
21     $L \leftarrow L - \{l_b\}$
22     $result = result \cup \{(A_i, l_b)\}$
23   end
24   return $result$
25 end
```

Fig. 5. The *Speculative Query Labeler* Algorithm

14–20). Finally, the best label for A_i is removed from the set o candidate labels (line 21), so that it is no longer considered for other attributes.

The analysis of this algorithm is straightforward. Let n be the number of attributes in the anonymous relation, m be the number of candidate labels, and k be the number of sample values taken. It is easy to see that, in the worst case, the algorithm runs in time $O(n \cdot m \cdot k)$. However, an interesting feature of our approach is that the algorithm can achieve high accuracy even when a small sample of data values are used. This is important because it reduces the number of speculative queries submitted in Line 13. (Indeed, Section 6 shows that our method achieves high accuracy with less than 10 sample values per attribute.) Obviously, the fewer speculative queries used for labeling an attribute, the better. In our experiments, the response time for running a speculative query using the Yahoo! Search Web Service[5] was less than 1.5 seconds on average, but with high variance.

6 Experimental Evaluation

We evaluated our method using 8 datasets from different application domains, with 52 anonymous attributes amongst them (see Table 4). All data used in our tests were obtained using automatic Web wrappers. Figure 6 gives sample tuples

[5] http://developer.yahoo.com/search/index.html

Table 4. Datasets used in our experiments

domain	Web site	tuples	attributes	labels
books	www.amazon.com	900	8	15
games	www.allgame.com	500	5	21
medicine	www.vitacost.com	615	7	22
movies	www.allmovie.com	700	5	20
music	www.allmusic.com	600	4	17
posters	www.postershop.com	510	8	14
teams	www.fifaworldcup.com	32	10	18
watches	www.watchzone.com	550	5	18

A_1	A_2	A_3	A_4	A_5
Romance	5 Stars	1965	Doctor Zhivago	David Lean
Comedy	5 Stars	1936	Modern Times	Charles Chaplin
Action	4.5 Stars	1981	Raiders of the Lost Ark	Steven Spielberg
Epic	4 Stars	1960	Spartacus	Stanley Kubrick

Sample tuples in the movies dataset. Manually extracted candidate Labels: actor, box office, certification, company, country, director, directed by, distributor, film, genre, language, movie, release, rating, rank, starring, theatrical run, title, votes, year.

A_1	A_2	A_3	A_4	A_5
Armani	AR5447	Ladies	Stainless steel bracelet	\$195.00
Seiko	SDWG32	Men	Stainless steel Butterfly clasp	\$400.00
Longines	L51580966	Petite	Stainless Steel Bracelet	\$1700.00
Casio	DW5600E1V	Men	Plastic, black	\$70.00

Sample of the watches dataset. Manually extracted candidate Labels: band, brand, category, condition, description, display, gender, features, material, movement, model, price, price range, savings amount, size, style, title, type.

Fig. 6. Samples of tuples from three of our datasets with the candidate labels

from two of the datasets. Our goal in selecting datasets was to maximize diversity; thus, we used real data from various domains, and we selected attributes with several distinct values, attributes with few distinct values, numerical attributes, text attributes, etc.

We report here on three experiments. First, we study the behavior of the label assignment algorithm with manually provided labels. Next, we study the effectiveness candidate selection method. Finally, we evaluate the two methods working together. All our accuracy results are given by comparing the labeling produced by our algorithm to a manually defined *correct* labeling. We used our best judgment for deciding when a label is adequate for a given attribute. All our experimental data, including the datasets, candidate labels, and the *correct*

Table 5. 1^{st} and 2^{nd} highest LAA values for all anonymous attributes. In all cases, the label with the highest LAA value is correct.

		A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
books	1^{st}	0.89	0.91	0.92	0.95	0.44	0.78	0.78	0.37	–	–
	2^{nd}	0.05	0.08	0.04	0.04	0.28	0.10	0.11	0.29		
games	1^{st}	0.74	0.34	0.35	0.58	0.29	–	–	–	–	–
	2^{nd}	0.07	0.33	0.26	0.13	0.22					
medicine	1^{st}	0.60	0.45	0.54	0.71	0.43	0.52	0.40	–	–	–
	2^{nd}	0.16	0.12	0.19	0.05	0.37	0.32	0.29			
movies	1^{st}	0.58	0.92	0.75	0.32	0.49	–	–	–	–	–
	2^{nd}	0.21	0.04	0.12	0.29	0.43					
music	1^{st}	0.55	0.63	0.89	0.67	–	–	–	–	–	–
	2^{nd}	0.13	0.24	0.03	0.11						
posters	1^{st}	0.47	0.50	0.63	0.77	0.68	0.92	0.76	0.67	–	–
	2^{nd}	0.11	0.18	0.11	0.12	0.12	0.06	0.13	0.12		
teams	1^{st}	0.64	0.43	0.59	0.64	0.49	0.75	0.91	0.73	0.57	0.41
	2^{nd}	0.15	0.18	0.27	0.19	0.38	0.06	0.02	0.05	0.33	0.34
watches	1^{st}	0.58	0.94	0.26	0.99	0.97	–	–	–	–	–
	2^{nd}	0.08	0.4	0.22	0.01	0.01					

labeling for each such dataset can be found at `http://www.dcc.ufam.edu.br/~msevalho/labeling`. All results reported in this paper were obtained using the Yahoo! Search Web Service.

6.1 Effectiveness of Speculative Labeling

We verify the effectiveness of our labeling algorithm using it with candidate labels manually selected from 10 distinct Web sites for each domain. (Figure 6 shows the candidate labels used in two of the tests.) We restrict ourselves to labels that are popular on the Web; that is, terms for which the number of documents containing them was greater than 1 million. We set the *a priori* preference for these labels, $P(l_i)$, to 1 for simplicity.

Table 5 shows the highest (1^{st}) and second highest (2^{nd}) average LAA values obtained in 50 runs of the algorithm (with 50 different samples of 3 tuples each), for all datasets. In all cases, the labels with the highest LAA are correct. Nevertheless, we have also found cases in which more than one label would be appropriate for a given attribute. This has happened, for instance, for attributes A_8 in books with labels list price (0.3659) and buy new (0.2855); A_3 in games with labels genre (0.3500) and category (0.2601); and A_4 in movies with labels film (0.3248) and movie (0.2929). In all cases, all labels seem equally plausible. Note that in most cases the highest LAA value is far greater than the second LAA value obtained for the same attribute.

6.2 Impact of Sample Size

This experiment studies the impact of the number of samples from the (active domain of the) anonymous datasets on the accuracy of the labeling. We repeated

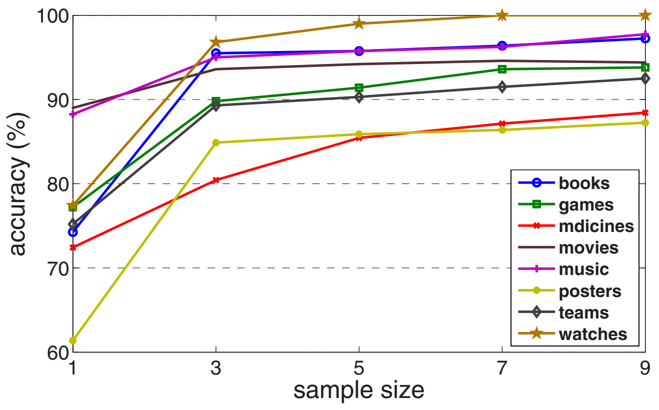


Fig. 7. Accuracy versus sample size for all datasets

the labeling process for each one of the 52 attributes using $k = 1, 3, 5, 7$ and 9 samples. For each value k, we repeated the labeling with 50 different samples. Figure 7 shows the accuracy of the method, defined as the fraction of the runs in which the label with highest LAA value corresponds to the correct label for the attribute. As the graph shows, 3 sample data values were enough to achieve an average accuracy around or above 90% for the datasets in the domains watches, music, books, movies, games and teams, which are fairly popular on the Web. For the medicine and posters datasets, which are less popular, the accuracy was above 80% using 3 samples.

It is worth noting that with datasets medicine and posters we observed much fewer responses to speculative queries than with the other ones. This could be explained by inaccuracies with some data values in our dataset (e.g., misspelled words) or by the fact that these datasets, as mentioned above, come from non-popular domains. Nevertheless, the results above indicate that our method can be very effective across domains even when one uses very few sample values from the anonymous dataset.

6.3 Cost of Speculative Labeling

We now discuss the cost of our labeling algorithm with respect to accessing the search engine. Table 6 shows the average number of speculative queries and the average execution time for the experiments reported in Section 6.2, per attribute and for different sample sizes. In addition, the average time spent with each individual speculative query was 1.4 seconds.

Note that to reach acceptable accuracy levels using 3 sample data values (Figure 7) required less than 50 speculative queries on average (per attribute), corresponding to about 1 minute of clock time. One may argue that this is a high cost compared to those methods that mine labels from Web pages. However, it must be noted that the time requirements in our method are comparable to most other data extraction tasks, such as fetching the data-rich pages, or generating

Table 6. Average number of queries (q) and average execution time in seconds (t), per anonymous attribute, for different sample sizes (k)

	$k=1$		$k=3$		$k=5$		$k=7$		$k=9$	
	q	t	q	t	q	t	q	t	q	t
books	12.75	22.71	34.50	55.21	56.13	85.96	79.00	111.89	111.13	142.00
games	20.27	21.77	57.00	66.86	95.00	98.50	129.25	169.88	169.55	245.85
medicine	20.41	29.60	61.24	94.30	99.54	142.34	139.36	199.28	179.17	238.18
movies	18.00	25.05	54.00	62.73	90.00	114.05	122.60	135.83	144.20	166.54
music	12.50	26.45	37.50	69.31	64.75	117.82	84.50	153.49	101.25	177.55
posters	10.50	16.74	31.93	42.82	48.85	56.84	68.29	74.36	81.56	116.59
teams	17.01	28.30	51.02	84.91	85.03	125.85	115.98	172.90	146.05	199.31
watches	16.60	24.90	50.78	62.49	80.00	97.24	110.63	138.29	132.19	159.81
aggregate	16.00	24.44	47.24	67.33	77.41	104.83	106.20	144.49	133.14	180.73

	$LMRR$	LAR	l
books	0.75	97.50%	60.4
games	0.53	86.00%	41.5
medicine	0.52	87.14%	47.4
movies	0.78	92.00%	37.9
music	0.64	97.22%	33.6
posters	0.37	88.75%	57.4
teams	0.82	98.00%	79.0
watches	0.72	96.00%	37.2

(a) Mean *LMRR*, *LAR*, and number of labels (l) per domain.

	accuracy	q	t
books	91.25%	141.70	139.47
games	84.00%	119.10	110.07
medicine	72.86%	139.70	132.06
movies	90.00%	108.30	105.12
music	92.50%	96.30	67.33
posters	63.75%	163.20	158.10
teams	90.00%	213.78	235.09
watches	82.00%	115.20	114.30

(b) Mean accuracy, number of queries (q), and execution time in seconds (t).

Fig. 8. Evaluation of the Candidate Labels Selection Method

wrappers. In light of the our high accuracy and the many limitations of the previous methods, our approach seems very attractive in practice.

6.4 Candidate Label Selection

This section evaluates the effectiveness of the candidate label selection method (recall Sections 3.1 and 4), which is aimed at producing a list of plausible labels for an anonymous dataset. The evaluation is based on two metrics, adapted from classical ones in Information Retrieval, which we discuss next.

Label MRR (LMRR). This metric is an adaptation of the well-known *Mean Reciprocal Ranking* metric (MRR). MRR is suitable for evaluating ranking functions, such as our label/attribute score, in contexts where the number of relevant elements in the ranking is expected to be small (such as in Q&A systems). Instead of measuring the rate of relevant elements in the ranking (i.e., precision), MRR measures how far down in the ranking is the first relevant answer.

For a given domain represented by an anonymous dataset $R(A_1, \ldots, A_n)$, we compute the LMRR according to Equation 2, where L_i is the ranking of the candidate labels with respect to attribute A_i (see Section 4) and $f(L_i)$ is the position of the uppermost plausible label for A_i in this ranking, according to a human judgment.

$$LMRR(R) = \frac{\sum_{A_i \in R} \frac{1}{f(L_i)}}{n} \tag{2}$$

Note that $LMRR$ values closer to 1 implies that a plausible label appears closer to the top of the ranking, and that the values are exponentially reduced as the position of the first plausible label in the ranking increases.

Label-Attribute Recall (LAR). This metric evaluates how well the labels in found by the label selection algorithm (denoted by L below) cover the attributes to be labeled. For a given domain represented by an anonymous dataset $R(A_1, \ldots, A_n)$, we compute the LAR as follows:

$$LAR(R) = \frac{m}{n} \quad , \tag{3}$$

where m is the number of attributes of R which have at least one plausible label in L.

The table in Figure 8(a) shows the mean $LMRR$ and LAR over 10 runs of the method over each domain. Because a different sample is used in each run (recall Section 4), it is possible that different candidate labels are found in different runs. Thus, table also shows the average number of candidate labels (l) obtained in the runs for each domain.

Figure 8(a) indicates that a plausible label is found before the third position on the ranking for all domains we have tested. This corroborates our assumptions regarding the co-occurrence of values and labels of a given attribute and shows that our score function for associating labels and attributes works well in practice. (This also confirms the fact that Hearst patterns alone are not enough for finding good labels, because the highest ranked label is not always the most appropriate one.) Regarding LAR, the table in Figure 8(a) shows that most of time time a plausible label is found for the anonymous attributes in all domains considered. In fact, even in the games domain, which has the lowest LAR (86%) in our tests, our method failed to find a suitable label only in 7 of the 50 trials (10 for each of the 5 attributes). In summary, the method was very effective in finding good labels for the anonymous attributes.

6.5 Label Selection and Assignment

This section evaluates the label selection method and the label assignment method working together. To do that, we run the labeling process for every attribute of all datasets using $k = 3$ samples (as suggested by the discussion in Section 6.2). The labeling process is repeated 10 times using the distinct sets of candidate labels generated by the label selection method in the previous experiment. The table in Figure 8(b) shows the average of accuracy values, the average number of speculative queries issued (q) and the average clock time in seconds (t), per attribute. The results are consistent with those using manually selected labels (Figure 7). For instance, note the relatively worse accuracy for domains posters and medicine. On the other hand, notice that the set of candidates labels used here is much larger than the set of manually selected ones (Figure 8(a)). As a result, the labelling process required more queries and time. Nevertheless, we believe these are very reasonable costs given the accuracy of the results.

7 Conclusion

We have proposed a novel and highly effective automated approach for labeling anonymous datasets extracted from the Web. By enriching the schemas of extracted data, labeling algorithms greatly help in data integration settings, which often rely on the existence of meaningful labels in the schemas [11]. The assignment of labels in our work is based on the probabilistic notion of *affinity* between (the domains of) anonymous attributes and candidate labels. We use the number of documents matched by *speculative queries* submitted to standard Web search engines for estimating the probabilities in our model. As demonstrated by extensive experimental results, our method is both very effective, indicated by the very high accuracy achieved, and efficient, indicated by the low average number of speculative queries needed before a labeling is found.

Unlike previous methods, our approach does not require the presence of labels in the Web content containing the data. By doing so, we allow user-defined labels to be considered as candidate labels, not restricting the user to accept those labels chosen by the content authors. We provide an effective algorithm for finding candidate labels for a given anonymous dataset that mines frequently occurring terms on Web documents which are used as hypernyms for some the values in the dataset.

Future Work. One immediate line of future work that we identify is exploiting existing repositories of Semantic Web metadata (e.g., Swoogle[6]) that index RDF [8] schemas and OWL [9] specifications as sources of candidate labels. Similarly to standard Web search engines, such repositories provide a keyword-based search interface allowing one to retrieve a ranked list of metadata documents (essentially, hierarchical schemas) that match given keywords. In our setting, this leaves open the question of which keywords to use for the search (note that the best search keywords are not necessarily data values, which is all we have as input in our setting). We are investigating the use of techniques for finding prominent keywords out of Web documents (e.g., see [10]) for this purpose.

An interesting observation from our experiments is that speculative queries on popular domains (e.g. books) return a much larger number of results compared to other less popular ones (e.g., watches). This raises interesting questions regarding the confidence in the labeling produced by our algorithm. First, we are interested in characterizing formally what would be an acceptable threshold for stopping the execution of speculative queries. Second, given that our algorithm samples from the anonymous dataset, we want to study how resilient it is to sampling bias.

[6] `http://swoogle.umbc.edu/`.

Acknowledgments. This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and from the Brazilian Research Council (CNPq) through projects GERINDO (CT-INFO 552.087/ 02-5), SIRIAA (CT-Amazônia 55.3126/2005-9) and through individual fellowships to A. S. da Silva (303032/2004-9) and J. M. B. Cavalcanti (303738/2006-5). D. Barbosa is also supported in part by an Ingenuity New Faculty award from the Alberta Ingenuity Fund.

References

1. Arasu, A., Garcia-Molina, H.: Extracting structured data from web pages. In: SIGMOD (2003)
2. Arlotta, L., Crescenzi, V., Mecca, G., Merialdo, P.: Automatic annotation of data extracted from large web sites. In: WebDB (2003)
3. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: Towards automatic data extraction from large web sites. In: VLDB (2001)
4. de Castro Reis, D., Golgher, P.B., da Silva, A.S., Laender, A.H.F.: Automatic web news extraction using tree edit distance. In: WWW (2004)
5. Etzioni, O., Cafarella, M., Downey, D., Shaked, A.-M.P.T., Soderland, S., Weld, D.S., Yates, A.: Unsupervised Named-Entity Extraction from the Web: An Experimental Study. Artif. Intell. 165(1), 91–134 (2005)
6. Hearst, M.A.: Automatic Acquisition of Hyponyms from Large Text Corpora. In: COLING, pp. 539–545 (1992)
7. Laender, A.H.F., Ribeiro-Neto, B.A., da Silva, A.S., Teixeira, J.S.: A brief survey of web data extraction tools. SIGMOD Record 31(2), 84–93 (2002)
8. Lassila, O., Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation (1999)
9. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview. W3C Recommendation (2004)
10. Rafiei, D., Mendelzon, A.O.: What is this page known for? Computing Web page reputations. Computer Networks 33(1-6), 823–835 (2000)
11. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 334–350 (2001)
12. Turney, P.D.: Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In: Flach, P.A., De Raedt, L. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, Springer, Heidelberg (2001)
13. Wang, J., Lochovsky, F.H.: Data extraction and label assignment for web databases. In: WWW (2003)
14. West, D.B.: Introduction to Graph Theory. Prentice-Hall, Englewood Cliffs (1996)
15. Wu, W., Doan, A., Yu, C.T.: WebIQ: Learning from the Web to Match Deep-Web Query Interfaces. In: ICDE (2006)
16. Zhai, Y., Liu, B.: Web data extraction based on partial tree alignment. In: WWW (2005)

Data Quality Enhancement of Databases Using Ontologies and Inductive Reasoning

Olivier Curé[1] and Robert Jeansoulin[2]

[1] S3IS Université Paris Est, Marne-la-Vallée, France
ocure@univ-mlv.fr
[2] IGM Université Paris Est, Marne-la-Vallée, France
robert.jeansoulin@univ-mlv.fr

Abstract. The objective of this paper is twofold: create domain ontologies by induction on source databases and enhance data quality features in relational databases using these ontologies. The proposed method consists of the following steps : (1) transforming domain specific controlled terminologies into Semantic Web compliant Description Logics, (2) associating new axioms to concepts of these ontologies based on inductive reasoning on source databases, and (3) providing domain experts with an ontology-based tool to enhance the data quality of source databases. This last step aggregates tuples using ontology concepts and checks the characteristics of those tuples with the concept's properties. We present a concrete example of this solution on a medical application using well-established drug related terminologies.

1 Introduction

The emergence of the Semantic Web and other semantic dependent applications encourages the design and use of ontologies. Although efficient and user-friendly ontology engineering tools (edition, visualization, storage, etc.) are now available, the design from scratch of domain ontologies is still considered a difficult and burdensome task. An approach generally adopted is that database schemata can support and thus ease, accelerate the design of expressive ontologies. Among these approaches, the most widely used aim to define a mapping between the source database schemata and a target ontology (see [20] as a survey of solutions on this topic).

In this paper, we propose another approach which takes advantages of the large number of databases maintained in the world as well as the many available hierarchical classifications, thesauri and taxonomies. In the ontology research field, these notions are associated with a set of concepts that are more or less strictly organized in hierarchies. The fundamental work of [6] analyzes the meaning of the taxonomic relationships and highlights that multiple types of taxonomic relationships exists. Also, as proposed in [14], depending on the context it is possible to interpret the hierarchical organizations of these terminologies as defining partial order relations on their concepts. In this paper, we concentrate on the contexts where such properties can be assumed and we therefore call them classifications.

The aim of our approach is twofold. In a first phase, we enrich classifications integrated in valuable databases using inductive reasoning (the output of this processing is henceforth called an ontology). In a second phase, we provide a graphical user interface which exploits ontologies created in phase one to detect and ease the repairing of inconsistencies in the source databases.

A main idea of our approach is to consider that the axioms added to the ontologies correspond to additional source database data dependencies. These data dependencies are strongly related to instances of the database and their representation are generally not supported in most standard relational database management systems (RDBMS). So this approach enables to store these data dependencies as valuable properties of expressive ontologies.

Starting from these resulting ontologies, we check if the source database violates some of these data dependencies and propose a data cleaning solution. Due to possible exceptions in the dataset, the automatic repairing of data may not be pertinent. Nevertheless, we assist the end-user by automatically detecting possible violations and by offering an effective and user-friendly graphical user interface to semi-automatically enable data repairing.

By repairing violations, we aim to enhance the data quality of the source databases. In this paper, we are being influenced by ISO 9000 Quality standards and consider two data quality elements : completeness and correctness. In terms of completeness, we are interested in the presence or absence of data in the dataset. We distinguish between two aspects of completeness : (i) comission, i.e. when excess data are present in the dataset, and (ii) omission, i.e. when some data are absent from the dataset. Considering the correctness aspect, we expect the dataset to contain the correct data.

This paper is organized as follows: in section 2, we present the basic notions involved in our data quality enhancing framework. In section 3, we highlight the motivating example of this research which is related to medical informatics and exploits drug databases. In section 4, the terminology enrichment using inductive reasoning approach is presented. Section 5 introduces our ontology-based detection and repairing solution and evaluates its efficiency on the drug database example. Section 6 emphasizes the adaptability of our solution to other domains such as geographical information. Some related works are proposed in Section 7 and Section 8 concludes the paper and emphasizes on some future work.

2 Basic Notions

This section reviews the main notions, related to relational databases and Description Logics (DL), needed to present our framework.

A fixed database schema is a finite sequence R=$\{R_1, .., R_n\}$ of relation symbols R_i, with $1 \leq i \leq n$, that are of fixed arity and correspond to the database relations. We define a relation R_i as a set of attributes $\{A_i^1, .., A_i^k\}$ where A_i^j, with $1 \leq j \leq k$, denote attributes over a possibly infinite domain D.

An *instance* I over a *schema* R is a sequence $(R_1^I, .., R_k^I)$ that associates each relation R_i with a relation of the same arity R_i^I. In this paper, we often abuse the notation and use R_i to denote both the relation symbol and the relation R_i^I that interprets it. We call a fact $R(t)$ the association between a tuple t of a fixed arity and a relation R of the same arity. If R is a *schema* then a dependency over R is a sentence in some logical formalism over R.

The ontologies are represented using a DL formalism [2]. This family of knowledge representation formalisms allows the represention and reasoning over domain knowledge in a formally and well-understood way. We assume readers are familiar with the semantics of DLs, though we recall that the syntax for concepts in $\mathcal{SHOIN}$(D) [15] are defined as follows, where C_i is a concept, A is an atomic concept, R is an object role, S is a simple object role, T is a datatype role, D is a datatype, o_i is an individual and n is a non-negative integer :

$C \rightarrow \mathrm{A} \mid \neg C_1 \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists\, \mathrm{R.C} \mid \forall\, \mathrm{R.C} \mid \geq \mathrm{n\ S} \mid \leq \mathrm{n\ S} \mid \{o_1, .., o_m\} \mid \geq \mathrm{n\ T} \mid \leq \mathrm{n\ T} \mid \exists\, \mathrm{T.D} \mid \forall T.D$

The reason of our interest in the $\mathcal{SHOIN}$(D) DL is its syntactical equivalence with the OWL DL language [11], an expressive ontology language developed by the W3C and which is already supported by numerous tools (editors, reasoners, etc.).

3 Motivating Example

The motivating example of our approach is related to the data quality assessment in a self-medication application [9] and a drug database. In a nutshell, our self-medication application enables patients to maintain medical information in a personal health care record, access data on mild clinical signs and over the counter drugs. In such a context, the data quality of the drug database is fundamental as incorrect or incomplete data, e.g. contra-indications, may exacerbate the health condition of the patient. The drug database stores, for all drugs sold in the french market, all the information available on the Summary of Product Characteristics (SPC) as well as some extra information such as a rating, based on an efficiency/tolerance ratio, and a comment from a team of health care professionals. Some of the most used classifications in the drug domain have also been integrated in our database and are used to link drug products to a certain concept in these classifications. In the rest of this section, we introduce the main characteristics of these drug classifications.

3.1 Drug Terminologies

Controlled terminologies and classifications are widely available for health care and bioinformatics [7]. In this paper, we are interested in the drug related classifications that are the most used in french drug databases, namely the European Pharmaceutical Market Research Association (EphMRA)© and the Anatomical Therapeutic Chemical (ATC) classifications. As most french drug databases, we use the CIP french codes to identify products. For instance, the drug *Tussidane*© syrup is sold as a 250ml bottle product identified with CIP value 3622682 and a 125ml bottle product with CIP value 3622676. So each product has a distinct CIP code and many CIPs may be available for a given drug, one for each product presentation.

EphMRA classification. The EphMRA brings together European, research-based pharmaceutical companies operating on a global perspective. One of the missions of the EphMRA is to provide recognised standards by continuously supporting and actively participating in establishing high levels of standards and quality control in pharmaceutical marketing research. The Anatomical Classification system (AC-system) is the main classification developed by the EphMRA, with its sister organisation in the USA, the Pharmaceutical Business Intelligence and Research Group (PBIRG). This system represents a subjective method of grouping certain pharmaceutical products. The products are classified according to their main therapeutic indication and each product is assigned to one category. In the AC-system, categories are organized on a cascade of 4 levels where each sub-level gives additional details about its upper-level.

The first level of the code is based on a letter for the anatomical group and defines 14 groups. The second level is used to regroup several classes together, in order to classify according to (i) indication, (ii) therapeutic substance group and (iii) anatomical system. This level adds a digit to the letter of the first level and enables the creation of the cascade classification. Therefore, before creating a new second level, all existing possibilities of classification should be analyzed. There could be cases where it is necessary to create a second level without a cascade to the third or fourth level. However, these cases are seldom in the current classification. The third level adds a letter to a second level code and describes a specific group of products within the second level. This specification can be a chemical structure or it can describe an indication or a method of action. The fourth level gives more details about the elements of the third level (formulation, chemical description, mode of action, etc.). Fourth level codes add a digit to third level ones.

The complete hierarchy for antitussive drugs corresponds to :

```
R: Respiratory system
 R5: Cough and cold preparations
   R5D: Antitussives
      R5D1: Plain antitussives
      R5D2: Antitussives in combinations
```

ATC classification. The ATC system [22] proposes an international classification of drugs and is part of WHO's initiatives to achieve universal access to needed drugs and rational use of drugs. In this classification, drugs are classified in groups at five different levels.

In fact, the ATC system modifies and extends the AC-system of EphMRA. Thus the first level is composed of the 14 groups of the EphMRA system. The second is also quite similar and corresponds to a pharmacological/therapeutic subgroup. The third and fourth levels are chemical/pharmacological/therapeutic subgroups. Finally, the fifth level corresponds to the chemical substances. With its fifth level, the ATC classification enables to classify drugs according to Recommended International Non-proprietary Names (rINN). This is different from EphMRA's classification where the leafs of the tree (fourth level) give details on a wider perspective (formulation chemical description, mode of action, etc.). Thus we consider that the use of both terminologies in our data quality assessment approach is complementary.

We now provide an extract from the ATC hierarchy for some cough suppressants:

```
R: Respiratory system
 R5: Cough and cold preparations
   R05D: Cough suppressants, excluding combinations
       with expectorants
     R05DA: Opium alkaloids and derivatives
       R05DA01 Ethylmorphine
       ...
       R05DA08 Pholcodine
       ...
       R05DA20 Combinations
```

The *R05DA20* code identifies compound chemical products that combine *opium alkaloids* with other substances. An example for this code is the *Hexapneumine*© syrup which contains the following chemical substance : *pholcodin*, *chlorphenamin* and *biclotymol* which respectively correspond to R05DA08, R06AB04 and R02AA20. In our approach, we argue on the relevance of classifying such products with the conjuction of their codes rather than with a single unifying code.

4 Terminology Enrichment Using Inductive Reasoning

The purpose of the terminology enrichment is to enable the aggregation of database tuples in a coherent way such that common properties can be discovered and associated to ontology concepts.

Intuitively, we consider that a relation, or a set of relations, $Term$ in the database schema R stores a given terminology, e.g. the ATC classification. Let consider that a relation Ind stores individuals of the database domain, e.g. drug products. Then it is most likely that a one-to-many relation, or a chain of relations, $TermInd$ relates facts between $Term$ and Ind. We can also assume that properties, e.g. contraindications or side-effects, about these individuals are either directly stored in Ind or stored in a relation $Prop$ in which case a possibly many-to-many relation $PropInd$ relates facts between $Prop$ and Ind. Thus the Ind relation plays a central role in our solution as it enables to join elements from $Term$ to elements of $Prop$. Given a fact in $Term$, it is possible to aggregate tuples from Ind, via $TermInd$, in a sufficiently coherent manner and to extract valuable properties from these groups. It is straightforward that such a query can be performed using SQL queries in most RDBMS.

4.1 Transformation into a Description Logic Formalism

In order to perform such enrichment, it is first necessary to transform the terminologies stored in *Term* into a DL formalism. This step has been performed using the DBOM [8] [10] Protégé plugin [13]. In a nutshell, DBOM (DataBase Ontology Mapping) enables to design and enrich existing OWL ontologies from relational databases by mapping relations to concepts and roles of the TBox via SQL queries. DBOM proposes a solution to the impedance mismatch problem between relational schemata and DL-based knowledge bases (KB) and supports the creation of simple and complex matching with declarative mappings. DBOM also enables the creation of ABoxes by processing the SQL queries associated to each mapped concepts and roles. Finally, this system also provides a preference-based approach to deal the inconsistencies.

For the terminology transformation operations, DBOM's input is the database schema R. The end-user defines mappings between relations of R and concepts of the ontology. The output is an OWL DL ontology [11]. In the context of our motivating example, the classification codes stored in *Term*, e.g. EphMRA or ATC codes, are transformed into OWL concepts. The classification levels are represented as a subsumption of concepts, i.e. using *rdfs:subClassOf* properties, and sibling concepts are declared disjoint, i.e. using *owl:disjointWith* properties.

In the following extract of our ATC ontology, we provide the description associated with the *Pholcodine* chemical substance concept, whose ATC code is *R05DA08*. On line 1, we can see that this concept is identified by a given URI with a local name corresponding to *R05DA08* ATC code. Line 2 defines the concept associated with the *R05DA* code (*Opium alkaloids and derivatives*) to be a super concept of this concept. On lines 3 and 4, we present examples of *disjointWith* properties between sibling concepts, only the first and last concepts are displayed for briefty reasons. Line 5 states a comment in the french language.

```
1. <owl:Class rdf:about="&p1;R05DA08">
2.  <rdfs:subClassOf rdf:resource="&p1;R05DA"/>
3.  <owl:disjointWith rdf:resource="&p1;R05DA01"/>
...
4.  <owl:disjointWith rdf:resource="&p1;R05DA20"/>
5.  <rdfs:comment xml:lang="fr">Pholcodine
6.    </rdfs:comment>
7. </owl:Class>
```

4.2 Enriching the Description Logic Via Induction

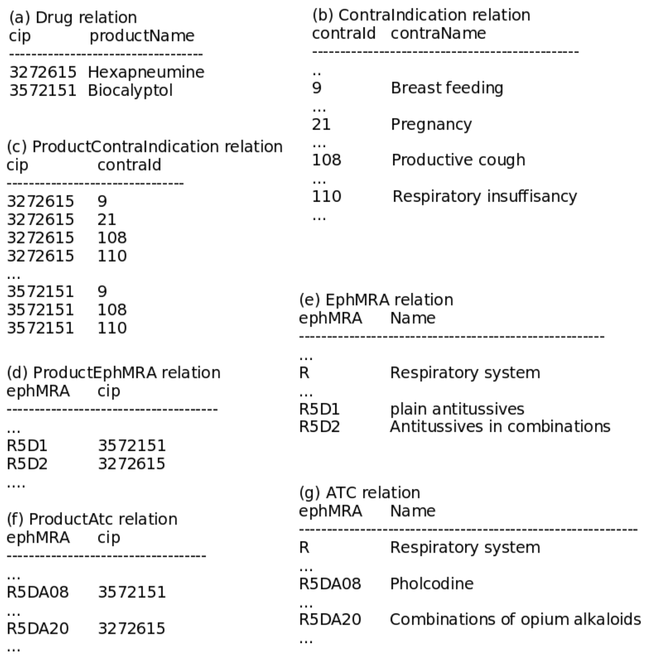
Starting from such an OWL ontology, which we now call $\mathcal{O}$, it is now possible to perform an enrichment by inductive reasoning on the information stored in the source database. Intuitively, the method presented exploits the one-to-many and many-to-many relations holding between the relations *Ind*, *Term* and *Prop* where tuples from *Ind* are first class citizen in the induction approach.

Given the relations $\{Term, Ind, TermInd, Prop, PropInd\}$ of a database instance R and the top concept ($\top$) of $\mathcal{O}$, the methods proceeds as follows. In a first step, it is necessary to select the relation *Prop* that leads the inductive process and to create an OWL concept associated to this relation, e.g. ContraIndication, and an object property, π, that relates concepts of $\mathcal{O}$ to this newly created concept, e.g. hasContraIndidication. We can now present the Induction-based ontology enrichment (IBOE) algorithm in which we consider that KB denotes the knowledge base being enriched, and a threshold θ that has been previously defined in the system. We consider that a binary relation *PropInd* (respectively *TermInd*) follows the pattern $\{A^1, A^2\}$, where these attributes correspond respectively to the primary keys in *Ind* and *Prop* (respectively $Term$ for $TermInd$). The input concept of IBOE is the starting concept from which we want to start the Domain ontology. It is generally assumed to start this process with the $\top$ concept.

Algorithm 1. Induction-based ontology enrichment (IBOE) algorithm
Input : concept c of the ontology
Output: enriched ontology $\mathcal{O}$

```
1.  for each sub concept sc of c do
2.    size = number of individuals in $Ind$ that are associated, via $TermInd$, to
      tuples of $Term$ related to sc
3.    if size>0 then
4.        for each {p,count} retrieved from the query SELECT $PropInd.A^2$,
      count(*) from $TermInd$,$PropInd$ WHERE $TermInd.A^2$ ilike 'sc%' AND
      $TermInd.A^1 = PropInd.A^1$ GROUP BY $PropInd.A^2$ HAVING count(*)/
      size $\geq \theta$ do;
5.        if no individual corresponding to p is in the KB then
6.            create individual(p)
7.            associate $\pi$ owl:hasValue individual(p) for the concept sc
8.        else
9.            if none of the superclasses of sc already has the property $\pi$ then
10.              retrieve individual(p) from KB
11.              associate $\pi$ owl:hasValue individual(p) for the concept sc
12.   IBOE(sc)
13. end do
```

In the context of our medical example, we provide with Figure 1 an extract from our drug database dedicated to medical contra-indications. Figure 1a proposes an extract, not all columns are displayed, of the *Drug* relation with two drugs and with their respective CIP identifiers. Figure 1b presents an incomplete list of the *ContraIndication* relation which stores all terms related to drug contra-indications. Now the *ProductContraIndication* relation enables to relate products identified by CIPs with their contra-indications (Figure 1c). Figure 1e provides an extract from the relation dedicated to the EphMRA classification (respectively Figure 1g for the ATC classification). Finally, two relations relate EphMRA and ATC codes to CIPs, respectively *ProductEphMRA* (Figure 1d) and *ProductATC* (Figure 1f) relations.



(a) Drug relation

cip	productName
3272615	Hexapneumine
3572151	Biocalyptol

(b) ContraIndication relation

contraId	contraName
..	
9	Breast feeding
...	
21	Pregnancy
...	
108	Productive cough
...	
110	Respiratory insuffisancy
...	

(c) ProductContraIndication relation

cip	contraId
3272615	9
3272615	21
3272615	108
3272615	110
...	
3572151	9
3572151	108
3572151	110

(d) ProductEphMRA relation

ephMRA	cip
...	
R5D1	3572151
R5D2	3272615
....	

(e) EphMRA relation

ephMRA	Name
...	
R	Respiratory system
...	
R5D1	plain antitussives
R5D2	Antitussives in combinations

(f) ProductAtc relation

ephMRA	cip
...	
R5DA08	3572151
...	
R5DA20	3272615
...	

(g) ATC relation

ephMRA	Name
R	Respiratory system
...	
R5DA08	Pholcodine
...	
R5DA20	Combinations of opium alkaloids
...	

Fig. 1. Extract of the XIMSA drug database

In the following, we present the inductive reasoning method on the EphMRA ontology, also named AC-ontology, and stress that an adaptation for the ATC ontology is straightforward. The method used to enrich the AC-ontology is based on induction reasoning on relevant groups of products, generated using the AC hierarchy. Intuitively, we navigate in the hierarchy of AC concepts and create groups of products for each level, using the ProductEphMRA relation, i.e. $TermInd$ on the IBOE algorithm. Then, for each group we study the information contained in the ContraIndication relation, i.e. $Prop$ in our database R and for each possible value in this domain we calculate the ratio of this value occurences on the total number of elements of the group. Table 1 proposes an extract of the results for the concepts of the respiratory system and the contra-indication domain. This table highlights that our self medication database contains 56 antitussives (identified by AC code *R05D*), which are divided into 44 plain antitussives products (*R05D1*) and 12 antitussives in combinations (*R05D2*). For the contra-indication identified by the number 76, i.e. allergy to one of the constituants of the product, we can see that a ratio of 1 has been calculated for the group composed of the *R* AC code. This means that all 152 products (100 %) of this group present this contra-indication. We can also stress that for this same group, the breast-feeding contra-indication (#9) has a ratio of 0.48, this means that only 72 products out the 152 of this group present this constraints.

Table 1. Analysis of contra-indications for the respiratory system

	R	R05	R05D	R05D1	R05D2
occurences	152	71	56	44	12
ContraId					
9	.48	.83	.86	.82	1
21	.26	.39	.3	.2	.73
76	1	1	1	1	1
108	.34	.69	.84	.84	.82
109	.35	.66	.8	.8	.82
110	.34	.73	.89	.86	1
112	.34	.71	.88	.86	.91
129	.4	.56	.8	.82	0.16

We now consider this ratio as a confidence value for a given AC-concept on the membership of a given domain's value. This membership is materialized in the ontology with the association of an AC-concept to a property, e.g. the hasContraIndication property, that has the value of the given contra-indication, e.g. breast-feeding (#9). In our approach, we only materialize memberships when the confidence values are greater or equal to a predefined threshold θ, in the contra-indication example this value is set to 0.6.

This membership is only related to the highest concept in the AC hierarchy and inherited by its sub-concepts. For instance, the breast feeding contra-indication (#9) is associated to the *R05* AC-concept as its confidence value (0.83) is the first column on the *contraId* 9 line that displays a confidence value greater or equal to θ (0.6) in the *R* hierarchy. Also, the pregnancy contra-indication (#21) is related to the *R05D2* AC concept since its value is (0.73).

Using this simple approach, we are able to enrich the AC-ontology with axioms related to several features of the SPC, e.g. contra-indication, side-effects, etc. At the end of this enrichment phase, the expressiveness of the newly generated ontology still corresponds to an OWL DL ontology. The following code proposes an extract of the AC-ontology, in RDF/XML syntax, where we can see the definition of *R05D2* concept (line #1 to #12). This description states that the concept :

– has the contra-indication identified by *CI_21* (line #2 to #7) which corresponds to pregnacy (line #13 to #16).
– is a subconcept of the *R05D* concept (line #8)
– is disjointWith the concept identified by the *R05D1* code
– has a comment, expressed in the french language (line #10).

```
1. <owl:Class rdf:about="&j.0;R05D2">
2.  <rdfs:subClassOf>
3.   <owl:Restriction>
4.    <owl:onProperty
          rdf:resource="&j.0;hascontraIndication"/>
```

```
    <owl:hasValue rdf:resource="&j.0;CI_21"/>
   </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="&j.0;R05D"/>
    <owl:disjointWith rdf:resource="&p1;R05D1"/>
   <rdfs:comment
       xml:lang="fr">ANTITUSSIFS EN ASSOCIATION
  </rdfs:comment>
</owl:Class>
<j.0:contraIndication rdf:about="&j.0;CI_21">
  <rdfs:comment xml:lang="fr">grossesse
  </rdfs:comment>
</j.0:contraIndication>
```

4.3 Ontology Refinement

In some situations, it may be necessary to revise the implantation of properties in the concept hierarchy. Table 2 highlights such a case if we consider that $x_2 \geq x_1 \geq \theta$ but $x_3 \leq \theta$ and concepts N_2 and N_3 are siblings and sub-concepts of N_1. The execution of the IBOE algorithm attaches property px to the N_1 concept as it is the first concept in the hierarchy that has a value, i.e. x_1, greater than θ. Table 2 emphasizes that the property is disbelieved for instances of the concept N_2 but it still holds for the concept N_2. Thus we consider that it is necessary to refine the attachment of px to the concept hierarchy. Figure 2 presents the refinement policy by changing the ontology from a state where px is associated to N_1 (Figure 2a) to a state where px is attached to all subconcepts of N_1 with a confidence value greater or equal to θ (Figure 2b), in this case only the N_2 node.

The line identified by contra-indication #129 in Table 1 highlights the need to refine the ontology. The resulting ontology has contra-indication #129 attached to the *R05D1* and not to *R05D* as originally deduced by the IBOE algorithm.

	N_1	N_2	N_3
px	x_1	x_2	x_3

Table 2. Confidence values

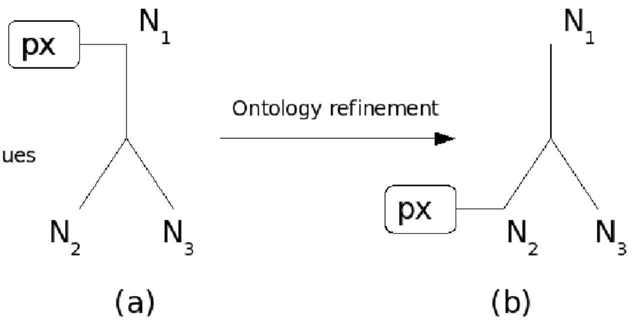


Fig. 2. Ontology refinement

This method can easily be applied to the ATC ontology or other drug related ontologies as soon as we consider that the ontology is presented in a DL formalism and a relation relates CIPs to identifiers of this ontology.

5 Ontology-Based Detection and Repairing

5.1 Detection Method

In this section, we only consider data quality violations at the completeness (comission and omission) and correcteness levels. The principle we use to detect these violations are supported by the ontologies defined in Section 4 and the relational database R, e.g. the drug database extract from Figure 1. The main assumption of this method is the following. We consider that the database R presents overall good data quality. This is the reason why we designed the ontology enrichment from induction on this database. But as data related dependencies are not supported in R, some data violations may exist in R. Thus we are using the properties associated to the concepts of our ontology(ies) to detect data quality violations. The potential of this approach is interesting because tuples from R can generally be aggregated using different characteristics, e.g. therapeutic class, chemical substances for the drug database. We also believe that an efficient approach to design relevant groups are based on the use of the terminologies that supported the design of these ontologies, e.g. EphMRA and the ATC terminologies.

We can view the relation between the ontologies and the relational database with a logical point of view. The schema part of a DL KB is typically called a TBox (terminology box) and is a finite set of universally quantified implications [2]. Most DLs can be considered as decidable fragments of first-order logic. Thus their axioms have an equivalent representation as first-order-formulae [5]. On the other hand, the schema of a relational database is defined in terms of relations and dependencies [16], also named integrity constraints. In [1], the authors explain that most dependencies can be represented as first-order formulae and have a dual role in relational databases : they describe the possible worlds as well as the states of the databases. Reiter also observed in [19] that integrity constraints are sentences about the state of the database and are not objective sentences about the world. As discussed in [17], although the expressivity of DLs underlying OWL and of relational dependencies is clearly different, the schema languages of the two are quite closely related.

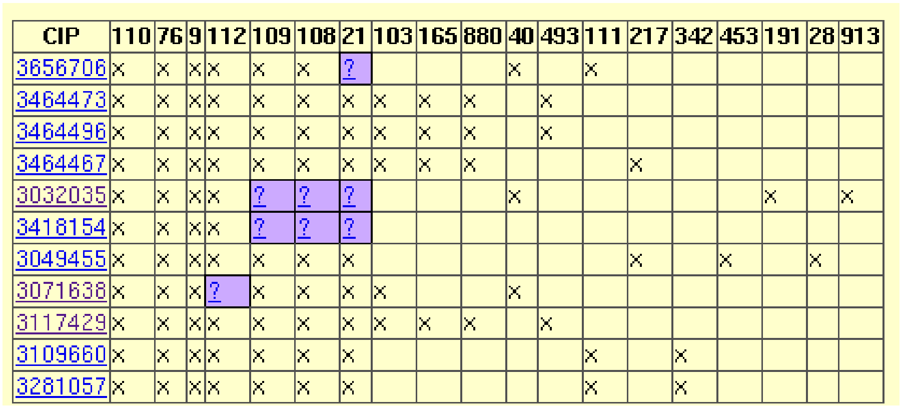
The interpretation of schemas in both DLs and relational databases are grounded in standard first-order semantics. In this semantics, a distinction is made between legal and illegal relational structures. A structure is legal when it satisfies all axioms defined in its schema, otherwise it is illegal. The terms used to denote legal structures in DLs and relational databases are different, respectively *models* and *database instances*.

Whenever a relational database is updated, its dependencies are interpreted as check. If the check is satisfied, the database instance is modified accordingly otherwise the update is rejected. The behavior on the models of DLs first-order formulae is different [12],[17].

In our approach, we consider that each database instances respect a given set of integrity constraints. But we also require from these database instances to respect a set of dependencies expressed in some associated ontologies, e.g. EphMRA and ATC ontologies in our medical example. Thus we want the database instances to satisfy explicitly provided database schema integrity constraints as well as the DL dependencies. The mechanism proposed for the latter has to deal with the context of a domain where many exception can occur, e.g. the pharmacology domain. For example, in the case of processing a group of drugs based on a given EphMRA concept, we may encounter drugs which do not present the same set of contra-indications. This may be caused by the dosage of the drugs, an aspect we are currently trying to solve with the integration of the DDD system, or the presence of excipients, an issue we aim to address with the integration of rules in the ontologies. In the current solution, in order to deal with these exceptions, we must involve users in the process of repairing violations. Thus our detect and repair approach is semi-automatic : detection is automatic and repairing involves the validation of the end-user. These steps are facilitated by the design of a web interface which highlights possible violations for a set of drugs and propose a fast and easy way to correct them.

We are actually proposing two graphical solutions to repair these violations: ontology concept-centric and database attribute-centric approaches.

Ontology concept-centric approach
In this approach two concepts from the ontology $\mathcal{O}$ are first selected. The first concept C_1 corresponds to a code of the $Term$ relation while the second one, C_2, corresponds to one of the concepts created from the relations $Prop$, e.g. the concept ContraIndication. The validation of this selection causes the display of a matrix where columns correspond to instances of the C_2 concept and the rows to database tuples, resulting from the execution of a system-generated SQL query, identified by a value x, i.e. drug products. At a row x and a column y, the matrice can be filled with 3 different values: (1) a 'x' symbol indicates that the tuple of Ind identified by value x has the property identified by value y for the C_2 concept, (2) a '?' highlights that the tuple does not have this property according to the data in the ontology, it should be the case, (3) an empty cell indicates that the drug does not have this property and this state holds with the ontology's knowledge. In case (2), the end-user can click on the interrogation mark to automatical correct the database by inserting a new tuple in $PropInd$ that states that the database individual has this property. Figure 3 proposes an extract from this presentation for the contra-indication fields for the *R05D2* EphMRA code. From such a matrix, it possible to click on a CIP number and to access the complete SPC of the drug. In this case, the composition field may help the health care professional to take a decision. Concerning possible database violations highlighted by interrogation marks in Figure 3, two violations are detected for the contra-indication #108 (*productive cough*): products identified



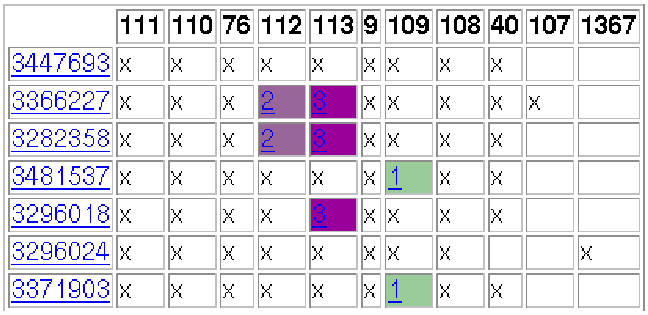
CIP	110	76	9	112	109	108	21	103	165	880	40	493	111	217	342	453	191	28	913
3656706	x	x	x	x	x	x	?				x		x						
3464473	x	x	x	x	x	x	x	x	x	x		x							
3464496	x	x	x	x	x	x	x	x	x	x		x							
3464467	x	x	x	x	x	x	x	x	x	x				x					
3032035	x	x	x	x	?	?	?				x						x		x
3418154	x	x	x	x	?	?	?												
3049455	x	x	x	x	x	x	x							x		x		x	
3071638	x	x	x	?	x	x	x	x			x								
3117429	x	x	x	x	x	x	x	x	x	x		x							
3109660	x	x	x	x	x	x	x						x		x				
3281057	x	x	x	x	x	x	x						x		x				

Fig. 3. Extract from an ontology concept-centric view: contra-indication matrix for the *R05D2* EphMRA concept

by CIPs 3032035 and 3418154. This detection is processed assuming that drugs of this category *R05D2* may all have the *productive cough* contra-indication. This aspect corrects data quality completeness related to comission. It is also possible to correct data quality completeness related to omission but deleting contra-indications in the SPC drug window.

Database attribute-centric approach
In a database attribute-centric approach an attribute of the database R, possibly not associated to an ontology concept, a concept created from a relation *Prop* and a set of available ontologies are selected. The selected attribute serves to design groups of database tuples and the set of ontologies enables to analyze this group according to the information stored in these ontologies. The results are displayed in a matrix similar to the one presented previously: columns are individuals of the concept C_2 and rows are tuples from the created group. The cells of the matrix can again be empty or filled with 'x' with the same interpretation as the previous approach. But the cells can also be filled with integer values that range from $2^n - 1$ with n the number of ontologies in the set. These values identify the elements of the powerset of n minus the empty set which is being dealt with the 'x' symbol. Figure 4 proposes an extract for the contra-indication SPC field for the *antitussive* therapeutic class. Both the EphMRA and ATC ontologies have been selected thus values range for 1 to 3:

- A value of 1 in a cell highlights a proposition made from the EphMRA ontology. This is the case for the contra-indication with value 109 and products identified with CIP 3481537 and 3371903.
- A value of 2 in a cell highlights a detection made from inferences using the ATC ontology.
- A value of 3 in a cell highlights that both ontologies (EphMRA and ATC) have detected this cell as a candidate for violation.



	111	110	76	112	113	9	109	108	40	107	1367
3447693	x	x	x	x	x	x	x	x	x		
3366227	x	x	x	2	3	x	x	x	x	x	
3282358	x	x	x	2	3	x	x	x	x		
3481537	x	x	x	x	x	x	1	x	x		
3296018	x	x	x	x	3	x	x	x	x		
3296024	x	x	x	x	x	x	x	x			x
3371903	x	x	x	x	x	x	1	x	x		

Fig. 4. Extract from a database attribute-centric view: contra-indication matrix for antitussive therapeutic class

5.2 Evaluation

We propose to evaluate this method on several aspects: (1) improvement of the resulting drug database after a thorough detection and repairing step, (2) satisfaction of the team of health care professionals maintaining the database. The evaluation emphasizes the results of the first execution of the detection and repairing process. This is the most relevant results as it is the step where the enhancements are most clearly visible. As database updates are performed, the data quality improvements are less evident but still as effective.

The first evaluation aspect of this method is to study the most prominent comparison criteria orginating from information retrieval: precision and recall. In logical terms, precision and recall correspond respectively to the correctness and completeness methods. Our evaluation studied the following SPC fields: contra-indications, drug interactions, cautions with allergies and diseases, cautions with other drug treatments, side-effects. The method we use proceeds as follows: during a detection and repairing session we evaluate the precision and recall for each row of our matrices. In such a setting, the precision is the measure of correctly found properties, 'x' in a matrix (true positives), over the total number of properties, the number of columns in the matrix (true and false positives). The recall measures the ratio of the true positives over the total number of properties for a row (true positives and true negatives). Testing over a set of more than 2000 drugs, we were able to evaluate the average precision and recall to respectively 0,71 and 0,61. Even more important is the estimation of improvements over both our criteria after our induction-based repairing. These improvements are calculated after the validation of a matrix by a domain experts, possibly with some repairings, e.g. the end-user modified the set of contra-indications for a given drug and thus changing the number of true positives and/or false positives. Over the same set of drugs, the ameliorations for precision was 6% and 8% for recall. This rate of improvement shows that the original database was of a relative high quality, which is good viewed from the inductive reasoning aspect. But it also shows that this database's data quality can be improved by our method. After each execution of the ontology-based detection and repairing, the ontology is refined by modification/validation of the domain experts responsible for this task. Finally, another interesting feature is the system's ability to check the data dependencies stored in the ontologies after each database updates.

The second aspect which is quite interesting is the satisfaction of our health care professionals. They consider that the detection and repairing method is really user-friendly, as it only involves reading and clicking, and the learning is quite short, as the detection only requires a visual detection. The best proof of success is the high usage rate of this maintenance assistant by our team of domain experts.

We believe that we can improve the end-user efficiency for the completeness comission factor which is less obvious and really involves a clear understanding of the grouping characteristic.

The efficiency of the detection method depends on the quality of the grouping factor. We believe that the system can assist end-users in selecting pertinent groups where the number of individuals is relevant. We can stress that the fewer products in the group, the easier it is to reach the membership threshold θ without really being relevant.

6 Application to Other Domains

There are other situations where the problem can be formulated into the same framework that has been used in this paper. This framework can be sketched this way:

– there exists some identifiable and observable item: here, an item is an individual "drug";
– there exists some properties, in some identifiable domain, that can be attached to the above mentionned observable item: here, the effects of each drug can be observed with respect to some contra-indication list, through series of more or less quantifiable observations;
– there exists a general taxonomy of individual components, whose lower level is made of individuals, which we can name atoms: here, an atom is a "chemical molecule";
– finally, there exists a partonomy of items in terms of atoms: here, this is the chemical composition of each drug as an association of molecules.

The enrichment procedure tries to propagate the appropriate properties from the atoms, up to the top of the hierarchy, pointing which contra-indications are most likely.

Let's consider the situation of landscape description and analysis:

– items are individual polygonal zones that we can observe, for instance from space;
– atoms are land parcels, stored as administrative or agricultural parcels, within some geographical database;
– the taxonomy of parcels is the hierarchy of space partitions at several levels (county, state, etc.) or any landscape sub-division;

– the partonomy describes the observed polygonal zones in terms of union of parcels (or intersecting parts);
– the observable properties can be crop classes, vegetation state, etc.

The enrichment procedure tries to propagate the appropriate properties from the atoms, up to the top of the hierarchy, describing which is the prominent land-use at any level, in terms of set of vegetation properties, etc. We can also introduce a second taxonomy on the observable properties, and propagate at each generalisation level, which is the most contributing part of space, in terms of set of space atoms.

Finally, the two enrichments, on the space taxonomy and on the property taxonomy, can be performed in a coordinate movement, and can be used for controlling the overal consistency. Even if this is propably a computational challenge, it is one of perpective for future investigation.

7 Related Work

The research work in data cleaning based on constraints are related to this work. A central aspect of these researches have been introduced in [3] where the notion of a repair is defined as the action to find another database that is consistent and minimally differs from the original database. In [3] repairs are provided by means of insertions and deletions while [23] proposes tuple updates as a repair primitive. Recently [4] introduced the notion of conditional functional dependencies (CFD) which is an extension of traditional functional dependencies and captures a fundamental part of the semantics of the data. The main objective of CFDs is to propose a new tool for data cleaning and toward this goal, [4] provides techniques and a sound and complete inference systems for their consistency and implication analyses. SQL-based techniques for detecting inconsistencies as violations of CFDs have been tested but the authors argue that much more work has to be done on several aspects, in particular on discovering CFDs. Like this work, our approach aims to store a new form of data dependencies to repair/clean existing databases but our approach focuses on storing these dependencies in DLs and much attention is given in terms of discovering the dependencies using inductive reasoning on the data.

The inductive approach has been used in the domain of Knowledge Discovery in Databases (KDD), in particular, to find classifications rules. Classification systems create mappings from data to predefined classes, based on features of the data. Many of the solutions adopting this approach are based on decision tree technology introduced in [18]. The obvious exploitation of this approach is to design a classification of concept based on data induction. Another relation between ontologies and such approaches is to exploit ontologies as background knowledge to enhance data mining applications [21]. So to our knowledge, this approach is a first step toward transforming existing classifications into expressive ontologies by inducing new concept properties.

8 Conclusion

We presented in this paper a simple yet effective and user-friendly solution to enhance the data quality of relational databases. In the process of repairing these databases, our solution enables to construct domain ontologies from available terminologies and classifications. We demonstrated this solution on the medical domain with drug databases. We believe that this method can be generalized to other domains where terminologies are accessible. In future, we aim to test our solution in the geographical domain with the Corine land cover hierarchy and the subsets of the United Nations Standard Products and Services Code (UNSPSC) products and services classification. Another extension we are working on is related to mapping ontologies using the inductive approach described. Actually, it is really easy to map the EphMRA and ATC ontologies because their concepts are both defined using the same set of terms, e.g. contra-indication, caution, side-effect terms. Finally, the extension that may have the biggest potential in the pharmaceutical domain is trying to automatize the generation of SPC from these and other ontologies. We think that the integration of the Defined Daily Dose (DDD) which is generally associated to the ATC in order to define drug posology.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley, Reading (1995)
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
3. Bertossi, L., Chomicki, J.: Query Answering in Inconsistent Databases Chapter in book. In: Chomicki, J., Saake, G., van der Meyden, R. (eds.) Logics for emerging applications of databases, Springer, Heidelberg (2003)
4. Bohannon, P., Fan, W., Geerts, F., Jia, X., Kementsietdsidis, A.: Conditional functional Dependencies for Data Cleaning
5. Borgida, A.: On the Relative Expressiveness of Description Logics and Precidate Logics. Artificial intelligence 82(1-2), 353–367 (1996)
6. Brachman, R.J.: What IS-A is and isn't: an analysis of taxonomic links in semantic networks. IEEE Computer 16, 30–36 (1983)
7. Cimino, J.J., Zhu, X.: The practical impact of ontologies on biomedical informatics IMIA Yearbook of Medical Informatics, pp. 1-12 (2006)
8. Curé, O., Squelbut, R.: A database trigger strategy to maintain knowledge bases developed via dat a migration. In: Bento, C., Cardoso, A., Dias, G. (eds.) EPIA 2005. LNCS (LNAI), vol. 3808, pp. 206–217. Springer, Heidelberg (2005)
9. Curé, O.: Ontology Interaction with a Patient Electronic Health Record. In: Proceedings of 18th IEEE Symposium on Computer-Based Medical Systems, pp. 185–190 (2005)
10. Curé, O., Squelbut, R.: Integrating data into an OWL Knowledge Base via the DBOM Protplug-in. In: Proceedings of the 9th International Protégé conference (2006)

11. Dean, M., Schreiber, G.: OWL Web Ontology Language Reference. W3C Recommendation (2004)
12. de Bruijn, J., Lara, R., Polleres, A., Fensel, D.: OWL DL vs. OWL flight: conceptual modeling and reasoning for the semantic Web. In: Proceedings of 14th international conference on World Wide Web, pp. 623–632 (2005)
13. Gennari, J., Musen, M., Fergerson, R., Grosso, W., Crubezy, M., Eriksson, H., Noy, N., Tu, S.: The evolution of protege: an environment for knowledge - based systems development. International Journal of Human - Computer Studies 123, 58–89 (2003)
14. Hepp, M., de Bruijn, J.: GenTax: a gerenric methodology for deriving OWL and RDF-S ontologies from hierarchical classifications thesauri, and inconsistent taxonomies. In: Proceedings of the European Semantic Web Conference (to appear, 2007)
15. Horrocks, I., Sattler, U.: A Tableaux Decision Procedure for SHOIQ. In: Proc. of IJCAI 2005, pp. 448–453 (2005)
16. Kanellakis, P.C.: Elements of relational database theory. In: Handbook of theoretical computer science (vol. B): formal models and semantics, pp. 1073–1156. MIT Press, Cambridge (1990)
17. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. In: Proceedings of the 16th International World Wide Web Conference, to appear (to appear 2007)
18. Quinlan, J.R.: Induction of Decision Trees. In: Readings in Machine Learning, pp. 81–106. Morgan Kaufamn, San Francisco (1990)
19. Reiter, R.: What Should a Database Know? Journal of Logic Programming 14(1-2), 127–153 (1992)
20. Shvaiko, P., Euzenat, J.: A Survey of Schema-Based Matching Approaches. Journal of Data Semantics IV, 146–171 (2005)
21. Taylor, M., Stoffel, K., Hendler, J.: Ontology-based Induction of High Level Classification Rules. Research Issues on Data Mining and Knowledge Discovery (DMKD) (1997)
22. WHO Collaborating Centre for Drug Statistics Methodology URL of Web site: http://www.whocc.no/atcddd/
23. Wijsen, J.: Condensed representation of database repairs for consistent query answering. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 378–393. Springer, Heidelberg (2002)

A Web Services-Based Annotation Application for Semantic Annotation of Highly Specialised Documents About the Field of Marketing

Mercedes Argüello Casteleiro[1],[*], Mukhtar Abusa[1], Maria Jesus Fernandez Prieto[2], Veronica Brookes[2], and Fonbeyin Henry Abanda[1]

[1] Research Institute of the Built and Human Environment (BUHU)
[2] School of Languages
The University of Salford, Salford, M5 4WT, United Kingdom
m.arguello@computer.org

Abstract. The field of marketing is ever-changing. Each shift in the focus of marketing may have an impact on the current terminology in-use, and therefore, compiling marketing terminology and knowledge can help marketing managers and scholars to keep track of the ongoing evolution in the field. However, processing highly specialised documents about a particular domain is a delicate and very time-consuming activity performed by domain experts and trained terminologists, and which can not be easily delegated to automatic tools. This paper presents a Web services-based application to automate the semantic annotation and text categorisation of highly specialised documents where domain knowledge is encoded as OWL domain ontology fragments that are used as the inputs and outputs of Web services. The approach presented outlines the use of OWL-S and the OWL´s XML presentation syntax to obtain Web services that easily deal with terminological background knowledge. To validate the proposal, the research has focused on expert-to-expert documents of the marketing field. The emphasis of the research approach presented is on the end-users (marketing experts and trained terminologists) who are not computer experts and not familiarized with Semantic Web technologies.

Keywords: Semantic Web, Semantic Web Services, Semantic annotation, Ontologies, OWL, OWL-S, XML, Text Categorisation.

1 Introduction

There is an increasing need to effectively mine for knowledge both across the Internet and in particular repositories. From the publishing industry to the competitive intelligence business, important volumes of data from various sources have to be processed daily and analysed by professional users [1]. In general, when expert professionals perform manual processing of specialised documents is to capture the pertinent

[*] Please note that the author has joined the *ESRC National Centre for e-Social Science (NCeSS)* based at the *University of Manchester*, M13 9PL, United Kingdom.

knowledge contained in each selected resource, and the knowledge captured is used to annotate the document by a set of descriptors (e.g. terms from a thesaurus). The introduction of tools that automatically extract/highlight specialised terminology from textual documents and are designed to be interactive and usable by domain experts and trained terminologists who are not necessarily computer experts, will contribute to save time from domain experts and trained terminologists and accelerate the process of semantic annotation of documents. Furthermore, make explicit the domain specific terminology used in a particular domain can be considered as "*feature extraction*" and may be used for *text clustering* and *text categorisation*.

Annotators were first conceived as tools that could be used to alleviate the burden of including ontology based annotations manually into Web pages [2]. Since then, many of them have evolved into more complete environments that use *Information Extraction* (IE) and *Machine Learning* (ML) techniques to propose semi-automatic annotations for Web documents. Nowadays, there are many annotation tools or environments such as Annotea [3], the Semantic Markup Tool [4], the OntoMat Annotizer [5], SMORE [6], SHOE Knowledge Annotator [7], ONTO-H [8]. However, some of them are not very well suited for the annotators unfamiliar with concepts related to ontologies and the semantic annotation in general. Moreover, most of the current annotation systems, like some of the ones mentioned-above, are applications that run locally on the annotator's computer.

The research approach presented in this paper is aligned with the annotation tool Saha [9], i.e. tackle the problem of creating semantically rich annotations by developing an annotation system that supports the distributed creation of metadata and that can be easily used by non-experts in the field of Semantic Web. However, the research approach presented in this paper differs from Saha [9] in:

1. The annotation application developed and presented in this paper is based on Web services [10] and considers three types of annotations: Dublin Core annotation, thesaurus annotation, and ontology-based annotation.
2. Background knowledge about specialised terminology is used to obtain a representation of documents as a selection of terms (term vectors) which is a pre-processing strategy where documents are reduced to terms considered "important", and therefore *text clustering* and *text categorisation* may profit from it. In fact, the Web services-based annotation application developed makes use of *text categorisation* to classify the content of highly specialised documents by means of relating parts of the document to one or several concepts of the domain ontology.
3. The research approach pays special attention to two tasks: the first task is to define the service's domain ontologies in terms of OWL [11] classes, properties, and instances. The second task is to create an OWL-S [12] description of the services, relating this description to the domain ontologies.
4. The research study addresses the challenge of *service composition* which refers to the process of combining different Web services to provide a *value-added* service [13]. The approach outlines the use of OWL-S [12] and the OWL´s XML presentation syntax [14] to obtain a combination of Web services that easily deal with terminological background knowledge.

This paper is organised as follows. Section 2 summarises related work. Requirements and an approach overview are in section 3. The details about the three different types of annotations are described in section 4. Section 5 presents the evaluation performed of the Web services-based annotation application in the marketing domain. Conclusions are in section 6.

2 Related Work

With the emergence of the Semantic Web, annotate metadata in documents and general Web resources have been the focus of many projects that have attempted to provide tools or frameworks for annotating different types of content (HTML, databases, multimedia) and with different degrees of automation – see `http://annotation.semanticweb.org/`. Some examples of widely used applications of metadata annotation will be the following:

1. **Dublin Core** [15]**:** is an example of a lightweight ontology that is being widely used to specify the characteristics of electronic documents without providing too many details about their content. It specifies a predefined set of document features such as *creator*, *date*, *contributor*, *description*, etc.
2. **Thesaurus and controlled vocabularies:** such as MeSH [16]. Terms from a thesaurus or from a controlled vocabulary can be used to provide agreed terms in specific domains and to annotate documents. Since these vocabularies are not completely formal, the annotations are normally pointers to those terms in the vocabulary [2].
3. **Ontologies:** such as the SWRC ontology [17]. The SWRC ontology generically models key entities in a typical research community and reflects one of the earliest attempts to put this usage of Semantic Web Technologies in academia into practice. The SWRC ontology initially grew out of the activities in the KA^2 project [18]. Since its initial versions it has been used and adapted in a number of different settings, most prominently for providing structured metadata for web portals, e.g. OntoWeb [19].

Dublin Core annotations are more ambiguous than annotations based on a thesaurus or controlled vocabulary, and these are also more ambiguous, in general, than the annotations based on ontologies [2]. However, these three approaches complement each other. To illustrate this: the most recent version of the SWRC ontology that has been released in OWL [11] format comprises a total of seven top level concepts namely *Document*, *Event*, *Organization*, *Person*, *Product*, *Project* and *Topic* and includes Dublin Core elements by means of both *datatype* and *object properties*. Furthermore, [20] proposes a method for transforming thesaurus into ontologies.

The foundations of the current approach are in line with the annotation tool Saha [9], i.e. tackle the problem of creating semantically rich annotations by developing an annotation system that supports the distributed creation of metadata and that can be easily used by non-experts in the field of Semantic Web. However, the current approach pursues to facilitate the economical annotation of large document collections. To achieve this, the current approach integrates terminological resources to facilitate the incorporation of knowledge extraction techniques into the annotation environment. Initially, terminological resources can be easily included in the SWRC ontology by means of a *Document Extension ontology* mainly because a controlled vocabulary, a dictionary or a thesaurus can be naturally seen as three different types of documents. The current approach pays special attention to thesaurus which are the most interesting terminological resources because a terminological entry in a thesaurus may contain synonyms, abbreviations, regional variants, definitions, contexts, even term equivalents in other languages. Different fields have thesaurus of their own which can be widely used for harmonising content indexing. For example, the MeSH [16] thesaurus is used to index the biomedical literature. Taking this into account, the current approach dedicated effort to obtain a suitable thesaurus, because with the help of thesaurus descriptors, domain experts can relate terminological entries from a thesaurus with class names from a classification system (e.g. concepts from a simple taxonomy ontology) to make explicit how terminological entries from a thesaurus cover a certain topic. The mapping process performed (i.e. mapping terms to concepts) has several advantages: a) the size of concept vectors representing documents is considerable smaller than the size of term vectors produced, and b) to facilitate providing the end-user with useful classifications of documents.

The combination of Web services [10] and ontologies [21] has resulted in the emergence of a new generation of Web services called *Semantic Web services* [22]. The landscape created by Semantic Web services has spurred several research issues. One important challenge is *service composition* which refers to the process of combining different Web services to provide a *value-added* service [13]. A large body of research has recently been devoted to Web service composition, and therefore, several techniques, prototypes, and standards have been proposed by the research community. However, these techniques, prototypes, and standards provide little or no support for the semantics of Web services, their messages, and interactions [10]. The research approach presented in this paper uses OWL-S [12] to describe Web services and explores the advantages of using the existing OWL Web Ontology Language XML Presentation Syntax [14] to encode OWL [11] domain ontology fragments as XML documents that are fruitful to be passed between Web services and that may be needed by other components in the same workflow. To validate the proposal, the research has focused on a Web services-based application which uses terminological knowledge to automate the semantic annotation and text categorisation of highly specialised documents (i.e. expert-to-expert documents). The next section provides an overview of the Web services-based annotation approach and requirements.

3 Requirements and Approach Overview

3.1 Requirements

In [23] seven requirements for semantic annotation systems have been formulated. A summary of how the current Web services-based annotation approach fulfils the requirements identified in [23] could be the following:

1. **Standard formats** - The Web Ontology Language (OWL) [11] has been used for representing ontologies, and a XML syntax based on a OWL's XML presentation syntax [14] has been used to pass ontology fragments between the services.

2. **User centred/collaborative design** - An easy to use interface that simplify the annotation process to end-users not familiarized with semantic Web techniques is achieved by means of a GUI partially generated on-fly by means of Ajax [24], where the existing XSLT stylesheet [25] and XML documents derived from the OWL's XML presentation syntax are interpreted by JavaScript functions that keep end-users unaware of underlying complexities.
3. **Ontology support** - Protégé 3.2 beta [26] has been chosen as the ontology-design and knowledge acquisition tool to: a) build ontologies in the Web Ontology Language OWL using the Protégé-OWL Plugin and b) to create OWL-S ontologies using the OWL-S Editor [27] that is implemented as a Protégé plugin.
4. **Support of heterogeneous document formats** -Documents can be in Web-native formats such as HTML. Although, MS Word format is also supported.
5. **Document evolution** - Documents may change as ontologies may change. This is even more likely in an environment where terminology plays pivotal role and may need to be up-dated on daily-basis. A way to keep ontologies and annotations consistent is to consider annotations as "temporary annotations" instead of "definitive annotations", and therefore, the environment should be able to generate or regenerate annotations automatically.
6. **Annotation storage** - The approach taken is in line with the Semantic Web model that assumes that annotations will be stored separately from the original document.
7. **Automation** - The level of automation achieved (generate or regenerate annotations automatically) guarantees the use by end-users without computer expertise.

3.2 Approach Overview

A Web service is a set of related functionalities that can be programmatically accessed through the Web [10]. A growing number of Web services are implemented and made available internally in an enterprise or externally for other users to invoke. These Web services can be reused and composed in order to realize larger and more complex business processes. The Web service proposals for description (WSDL[28]), invocation (SOAP[29]) and composition (WS-BPEL [30]) that are most commonly used, lack proper semantic description of services. This makes hard to find appropriate services because a large number of syntactically described services need to be manually interpreted to see if they can perform the desired task. Semantically described Web services make it possible to improve the precision of the search for existing services and to automate the composition of services. Semantic Web Services (SWS) [22] take up on this idea, introducing ontologies to describe, on the one hand, the concepts in the service's domain (e.g. flights and hotels, tourism, e-business), and on the other hand, characteristics of the services themselves (e.g. control flow, data flow) and their relationships to the domain ontologies (via inputs and outputs, preconditions and effects, and so on) [27]. Two recent proposals have gained a lot of attentions: 1) the American-based OWL Services (OWL-S) [12] and 2) the European-based Web Services Modelling Language (WSML) [31]. These emerging specifications overlap in some parts and are complementary in other parts. WSML uses its own lexical notation, while OWL-S is XML-based.

The OWL Web Ontology Language for Services (OWL-S) [12] provides developers with a strong language to describe the properties and capabilities of Web Services in such a way that the descriptions can be interpreted by a computer system in an automated manner. The current approach pays special attention to the *Service Process Model* because it includes information about inputs, outputs, preconditions, and results and describes the execution of a Web service in detail by specifying the flow of data and control between the particular methods of a Web service. The execution graph of a Service Process Model can be composed using different types of processes and control constructs. OWL-S defines three classes of processes. Atomic processes (`AtomicProcess`) are directly executable and contain no further sub-processes. From the point of view of the caller atomic processes are executed in a single step which corresponds to the invocation of a Web service method. Simple processes (`SimpleProcess`) are not executable. They are used to specify abstract views of concrete processes. Composite processes (`CompositeProcess`) are specified through composition of atomic, simple and composite processes recursively by referring to control constructs (`ControlConstruct`) using the property *ComposeOf.* Control constructs define specific execution orderings on the contained processes.

The research study addresses the challenge of *service composition* which refers to the process of combining different Web services to provide a *value-added* service [13]. The approach highlights the benefits of Semantic Web technologies in order to obtain a combination of Web services that easily deal with terminological background knowledge to automate the semantic annotation and text categorisation of highly specialised documents. The current research study exposes the advantages of using the existing OWL's XML Presentation Syntax [14] to encode OWL [11] domain ontology fragments as XML documents that are fruitful to be passed between Web services and that may be needed by other components in the same workflow. The current approach considers two Web services:

1. *Taxonomy-Thesaurus mapping service*: is a service that provides functionality to associate terminological entries from a thesaurus with concepts from a simple taxonomy ontology to make explicit how terminological entries from a thesaurus cover a certain topic. The *Taxonomy-Thesaurus* mapping performed plays pivotal role to obtain a *text classifier* that could carry out automatic *text categorisation.*
2. *Semantic annotation service*: is a service that provides functionality to perform three different types of annotations from textual documents: a) Dublin Core annotation, b) thesaurus annotation, and c) ontology-based annotation which is devoted to describe the content of documents, and where *thematic* metadata is used to describe the semantics of the document.

Each service considers different kinds of activities. It is necessary to detail each activity and consider if the activity can be related to an atomic process or to a composite process that can be further refined into a combination of atomic processes. Furthermore, it is essential to decide what are the inputs and outputs for each of the considered processes. Figure 1 and 2 show the inputs and outputs for composite processes of the *Taxonomy-Thesaurus mapping service* and the *Semantic annotation service* respectively. The name of each input or output is specified in (bold black) as well as a type is defined for each input and output (between brackets in grey). Inputs' and outputs' types are classes/concepts of ontologies that appear in figure 3.

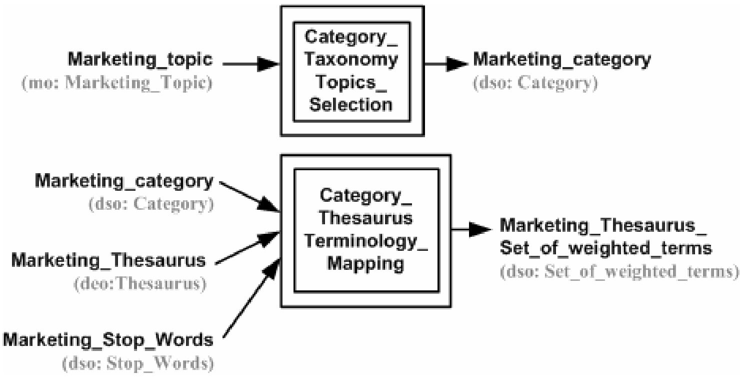


Fig. 1. Inputs and outputs for composite processes of the *Taxonomy-Thesaurus mapping service*

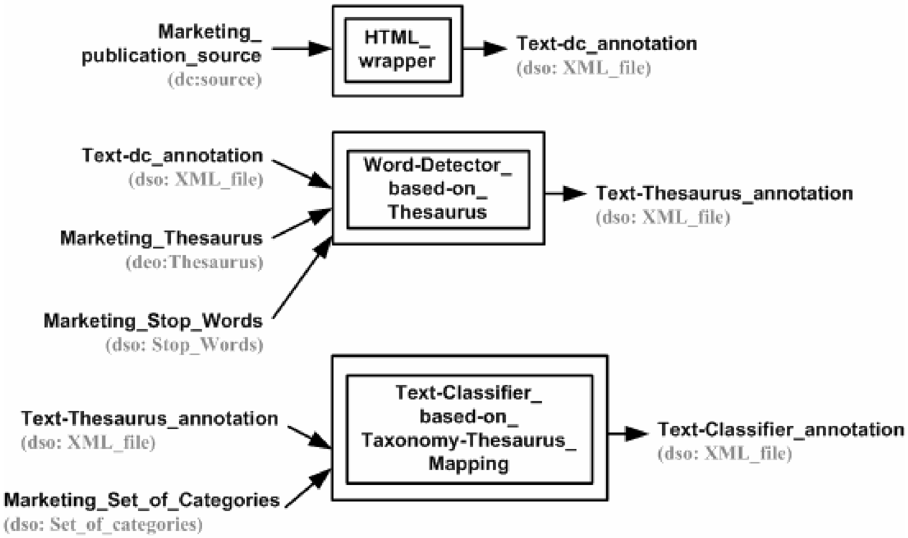


Fig. 2. Inputs and outputs for composite processes of the *Semantic annotation service*

The research study presented in this paper is adhered to a modular ontology design. Existing methodologies and practical ontology development experiences have in common that they start from the identification of the purpose of the ontology and the need for domain knowledge acquisition [32], although they differ in their focus and steps to be taken. In this study, the three basic stages of the knowledge engineering methodology of CommonKADS [33] coupled with a modularised ontology design have been followed:

I. KNOWLEDGE IDENTIFICATION: in this first stage, several activities were included: explore all domain information sources in order to elaborate the most complete characterisation of the application domain, and list potential components for reusing. The following knowledge sources were identified: a) the *SWRC ontology* [17] which generically models key entities relevant for typical research communities and the relationships between them, b) a *taxonomy of marketing topics* that appears in [34], and c) several terminological resources related to the marketing field, such as [35] or [36].

II. KNOWLEDGE SPECIFICATION: in this second stage, the domain model was developed. An overview of the modular ontological design appears in figure 3.

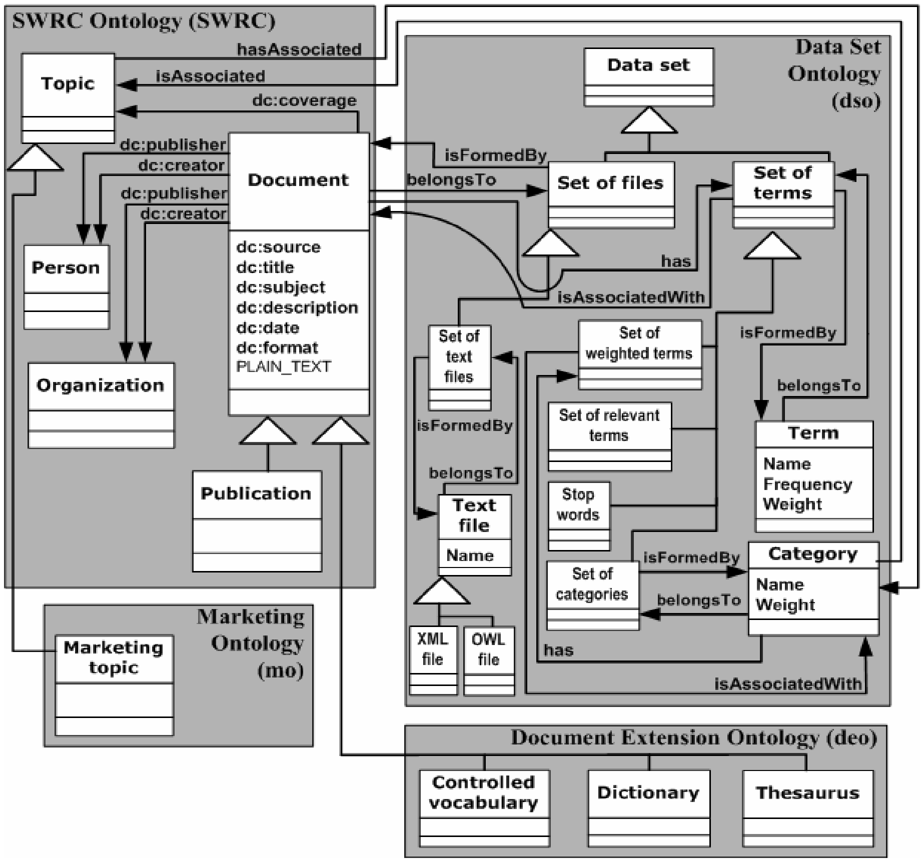
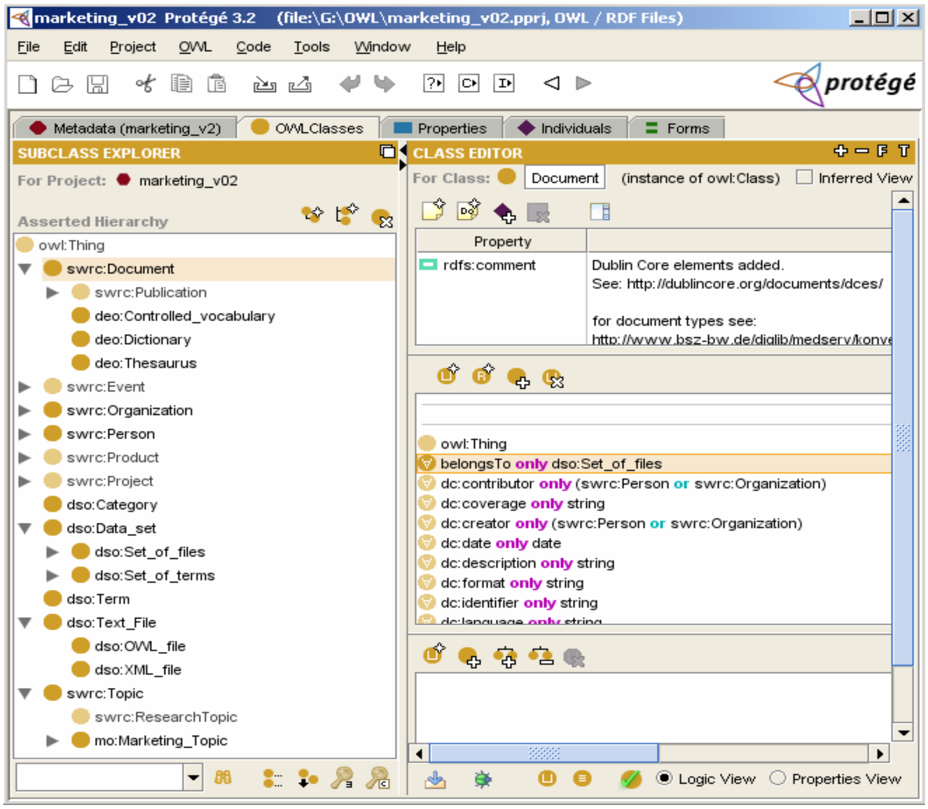


Fig. 3. Overview of the modular ontological design

Four ontologies have been considered: 1) the *SWRC ontology* [17] where several top level concepts and relationships have been reused, 2) the *Document Extension ontology* which is an extension of the SWRC ontology to include terminological resources, 3) the *Marketing ontology* which can be considered as an extension of the SWRC ontology to incorporate an adaptation of the *taxonomy of marketing topics* from [34], and 4) the *Data Set ontology* which is introduced to facilitate the linkage between inputs' and outputs' types of Web services and classes/concepts of the other three ontologies. Protégé 3.2 beta [26] has been chosen as the ontology-design and knowledge acquisition tool to build OWL [11] ontologies. Figure 4 shows a screenshot of Protégé 3.2 beta during the OWL ontology development that illustrates the relationship between the *SWRC ontology* and the *Data Set Ontology (dso)* by means of the *object property belongsTo* which is remarked.


marketing_v02 Protégé 3.2 (file:\G:\OWL\marketing_v02.pprj, OWL / RDF Files)
File Edit Project OWL Code Tools Window Help
protégé
Metadata (marketing_v2) OWLClasses Properties Individuals Forms
SUBCLASS EXPLORER
CLASS EDITOR
For Project: marketing_v02
Asserted Hierarchy
For Class: Document (instance of owl:Class) Inferred View
Property
Logic View Properties View

owl:Thing
swrc:Document
swrc:Publication
deo:Controlled_vocabulary
deo:Dictionary
deo:Thesaurus
swrc:Event
swrc:Organization
swrc:Person
swrc:Product
swrc:Project
dso:Category
dso:Data_set
dso:Set_of_files
dso:Set_of_terms
dso:Term
dso:Text_File
dso:OWL_file
dso:XML_file
swrc:Topic
swrc:ResearchTopic
mo:Marketing_Topic

rdfs:comment — Dublin Core elements added.
See: http://dublincore.org/documents/dces/

for document types see:
http://www.bsz-bw.de/diglib/medserv/konve

owl:Thing
belongsTo **only** dso:Set_of_files
dc:contributor **only** (swrc:Person **or** swrc:Organization)
dc:coverage **only** string
dc:creator **only** (swrc:Person **or** swrc:Organization)
dc:date **only** date
dc:description **only** string
dc:format **only** string
dc:identifier **only** string
dc:language **only** string

Fig. 4. A screenshot of Protégé 3.2 beta during the OWL ontology development

III. KNOWLEDGE REFINEMENT: in this third stage, the resulting domain model is validated by paper-based simulation, and more terms from [36] are added to the marketing thesaurus developed. It is also evaluated how each terminological entry added to the thesaurus is associated to one or more categories of a finite set of categories (selection of concepts from the taxonomy of marketing topics).

The next section provides details about the three different types of annotations considered. The annotation process relies on a combination of Web services. To enable a Web services composition that easily deal with terminological background knowledge, the research approach relies on OWL-S [12] to describe Web services and exposes the advantages of using the existing OWL's XML presentation syntax [14] to encode OWL [11] domain ontology fragments as XML documents that are fruitful to be passed between Web services and that may be needed by other components in the same workflow.

4 Annotation

Services can be described as a collection of atomic or composite processes, which can be connected together in various ways, and the data and control flow can be specified. Figure 5 shows the control flow and data flow for the three composite processes of the *Semantic annotation service*. The details about how to encode OWL ontology fragments as XML documents that are fruitful to be passed between processes in the same workflow are described below.

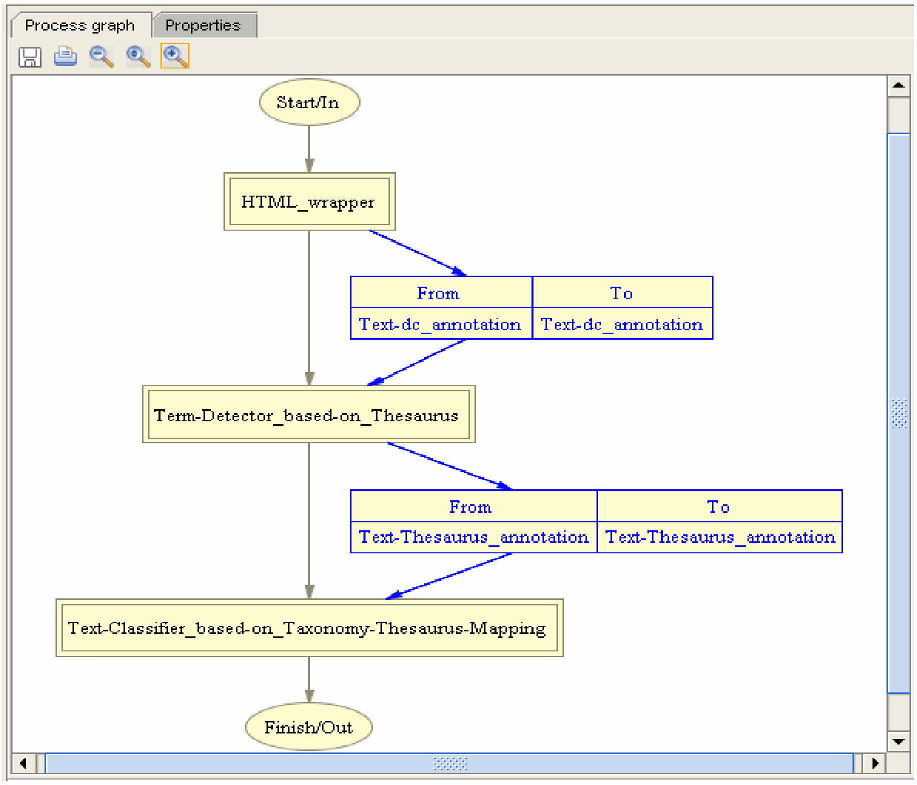

Process graph
Properties
Start/In
HTML_wrapper
From
To
Text-dc_annotation
Text-dc_annotation
Term-Detector_based-on_Thesaurus
From
To
Text-Thesaurus_annotation
Text-Thesaurus_annotation
Text-Classifier_based-on_Taxonomy-Thesaurus-Mapping
Finish/Out

Fig. 5. Control flow and data flow for composite processes of the *Semantic annotation service*

Web services are part of a trend in XML-based distributed computing and XML does not provide any means of talking about the semantics (meaning) of data. However, a XML document type definition can be derived from a given ontology as pointed out in [37]. The linkage has the advantage that the XML document structure is grounded on a true semantic basis.

There is more than one way to derive an XML Schema [38] from an OWL ontology which is compatible with the RDF/XML syntax. Many possible XML encodings could be imagined, but the most obvious solution is to use the existing OWL Web Ontology Language XML Presentation Syntax [14] which is the solution taken here. The owlx namespace prefix should be treated as being bound to `http://www.w3.org/2003/05/owl-xml`, and is used for the existing OWL's XML presentation syntax. The subsection 4.3 provides an example of individual axioms (also called "facts") based on the XML presentation syntax for OWL. These *facts* are outputs and/or inputs for each of the three composite processes of the *Semantic annotation service* that appears in figure 5.

4.1 Dublin Core Annotation

Documents may be in many different formats. The current approach considers two document formats: HTML and MS Word. Depending on the document source provided, a *HTML-wrapper* (see figure 5) or a *MS Word-text converter* is being used. Because, the world-wide-web (i.e. WWW) has become one of the most widely used information resources, the *HTML-wrapper* has being proved as more useful for automatic processing.

The *HTML-wrapper*, which appears in figure 5, performs the task of extract the data embedded in Web pages for further processing. The goal of the *HTML-wrapper* is to translate the relevant data embedded in Web pages into a structured format: the Dublin Core [15] lightweight ontology that is being widely used to specify a predefined set of document features such as *title*, *creator*, etc. The *HTML-wrapper* performs a *Dublin Core annotation* to create as an output a XML document (`Text-dc_annotation`) that will contain a set of document features according with the Dublin Core lightweight ontology and will be used as the input of the *Term-Detector_based-on_Thesaurus* (see figure 5).

4.2 Thesaurus Annotation

The *Term-Detector_based-on_Thesaurus* (see figure 5) uses simple terms and/or compound terms from thesaurus of specific domains to annotate documents. The process could be seen as an automatic indexing with controlled vocabularies/ thesaurus, and therefore, is closely related to automated metadata generation.

Terminological resources are increasingly available on-line, e.g. TERMIUM [39] or EURODICAUTOM [40]. In the particular case of the marketing field, many online dictionaries and glossaries of marketing terms could be found. However, not all of them appear to be comprehensive enough. To illustrate this: on the one hand, the *American Marketing Association* offers an on-line marketing dictionary [35] which is a good resource of information, although well researched definitions are not always provided. On the other hand, the *Faculty of Business and Economics* at the *Monash*

University (Australia) offers an on-line marketing dictionary [36] which is a comprehensive glossary of marketing terms that offers well researched definitions for most of the marketing related terms likely to be needed on this topic, and which could be easily converted into a thesaurus where each terminological entry may contain synonyms, abbreviations, regional variants, and definitions.

Text categorisation is one of the core problems in *Text Mining*. The goal of *text categorisation* is to automatically assign text documents to a finite set of predefined categories. With the rapid growth of Web pages on the World Wide Web (WWW), text categorisation has become more and more important in both the world of research and applications. One important challenge for large-scale *text categorisation* is how to reduce the number of *features* that are required for building reliable text classification models. There are typically two types of algorithms to represent the feature space used in classification. One type is the so-called "*feature selection*" algorithms, i.e. to select a subset of most representative features from the original feature space. Another type is called "*feature extraction*", i.e. to transform the original space to a smaller feature space to reduce the dimension.

The use of thesaurus can be seen as a type of "*feature selection*" because only the terms which belong to each terminology entry of the thesaurus will be taken into account. Furthermore, the use of thesaurus is expected to contribute to "*feature extraction*" as well, because most probably only a portion of the terms from the thesaurus will be found in the text documents of the corpus considered.

The *Dublin Core annotation* performed by the *HTML-wrapper* is more ambiguous than the annotations based on a thesaurus or controlled vocabulary. As depicted in figure 5, the XML document (`Text-dc_annotation`), which is the output of the *HTML-wrapper*, is complemented by the *Term-Detector_based-on_Thesaurus* which performs a *thesaurus annotation* to create as an output a XML document (`Text-Thesaurus_annotation`) that introduces a local (document level) measure: term frequency. For each simple or compound term t_k from thesaurus that appears in the document d_j, it is annotated not only the name of the term but also the term frequency (term counts) tf_k, i.e. the number of times that a term t_k occurs in a document d_j.

4.3 Ontology-Based Annotation

The *Text-Classifier_based-on_Taxonomy-Thesaurus-Mapping* (see figure 5) performs the task of annotate the document with *thematic* metadata by relating parts of the document (simple terms and/or compound terms previously identified that belongs to a thesaurus) to one or several concepts of the domain ontology (e.g. concepts from a taxonomy of topics).

The construction of the *Text-Classifier_based-on_Taxonomy-Thesaurus-Mapping* involves: a) a phase of *term selection*, in which the most relevant terms for the classification task are defined and where simple and compound terms which belong to terminological entries of thesaurus have been selected, and b) a phase of *term weighting*, in which weights for the selected terms are computed based on the explicit associations performed between terminological entries from thesaurus with concepts from a simple taxonomy ontology.

Thanks to the *Taxonomy-Thesaurus mapping service* described in subsection 3.2, the terminological entries of a thesaurus can be grouped and assigned to a finite set of categories (selection of concepts from the domain ontology), and therefore, the text indexing performed can be considered as an instance of *text categorisation.*

Text categorisation problems are usually *multi-class* in the sense that there are usually more than two possible categories. Although in some applications there may be a very large number of categories, the current research study focuses on the case in which there are a small to moderate number of categories. It is also common for *text categorisation* tasks to be *multi-label*, meaning that the categories are not mutually exclusive so that the same document may be relevant to more than one category.

In the particular case of the marketing field, there is a high overlapping between categories. Ranking categorisation has been introduced to deal with overlapping categories. In other words, for a given document d_j, the existing categories are ranked according to their estimated appropriateness to d_j, without taking any "hard" decision about any of them.

As pointed out in [41] the inductive construction of a ranking classifier for category $c_i \in C$ usually consists in the definition of a function $CSV_i: D \rightarrow [0,1]$ that, given a document d_j, returns a *categorization status value* for it, that is, a number between 0 and 1 which, roughly speaking, represents the evidence for the fact that $d_j \in c_i$. The CSV_i function takes up different meanings according to the learning method used. For example, probabilistic classifiers (see [42] for a thorough discussion) view $CSV_i(d_j)$ in terms of a probability.

The *thesaurus annotation* performed by the *Term-Detector_based-on_Thesaurus* is, in general, more ambiguous than the annotations based on ontologies. As depicted in figure 5, the XML document (`Text-Thesaurus_annotation`), which is the output of the *Term-Detector_based-on_Thesaurus*, is complemented by the *Text-Classifier_based-on_Taxonomy-Thesaurus-Mapping* which performs an *ontology-based annotation* to create as an output a XML document (`Text-Classifier_annotation`). An example of *ontology-based annotation* appears in figure 6 where significant *facts* from a XML document (`Text-Classifier_annotation`) have been included. The research approach presented in this paper estimates the appropriateness of a category c_i to a document d_j based on the weight of a category c_i in a document d_j, wc_{ij}, which is determined by a combination of a local (document level) measure and a global (thesaurus level) measure. The local (document level) measure is the term frequency (term counts) tf_k, i.e. the number of times that a simple term or a compound term t_k from a thesaurus appears in the document d_j. The global (thesaurus level) measure is the term weight w_k,i.e. the value assigned to a simple or compound term t_k from a thesaurus. The weighting scheme considers: on the one hand, that terms from a thesaurus that have been associated to too many categories (high overlapping) should receive a low weight, while terms from a thesaurus that have been associated to only one category should receive a high weight. On the other hand, the total number of terms from a thesaurus that have been associated to each category should be taken into account to prevent that categories with higher number of associated terms will become predominate.

```
<owlx:Individual>
<owlx:type owlx:name="Article">
. . .
<owlx:ObjectPropertyValue owlx:property="has">
<owlx:Individual>
<owlx:type owlx:name="Set_of_categories">
<owlx:ObjectPropertyValue owlx:property="isFormedBy">
<owlx:Individual>
<owlx:type owlx:name="Category">
<owlx:DataPropertyValue owlx:property="name">
<owlx:DataValue owlx:datatype="&xsd;string">
MC_Customer_and_Marketing</owlx:DataValue>
</owlx:DataPropertyValue>
<owlx:DataPropertyValue owlx:property="weight">
<owlx:DataValue owlx:datatype="&xsd;string">
464.99</owlx:DataValue>
</owlx:DataPropertyValue>
<owlx:ObjectPropertyValue owlx:property="isAssociated">
<owlx:Individual>
<owlx:type owlx:name="Marketing_topic">
<owlx:DataPropertyValue owlx:property="name">
<owlx:DataValue owlx:datatype="&xsd;string">
Customer_and_Marketing</owlx:DataValue>
</owlx:DataPropertyValue>
</owlx:Individual>
</owlx:ObjectPropertyValue>
. . .
</owlx:Individual>
</owlx:ObjectPropertyValue>
. . .
</owlx:Individual>
</owlx:ObjectPropertyValue>
. . .
</owlx:Individual>
```

Fig. 6. Significant *facts* related to *ontology-based annotation*

5 Evaluation

The evaluation of the Web services-based annotation application considers: 1) the experimental results obtained over two data sets of highly specialised documents (i.e. expert-to-expert documents) that were selected among the vast marketing corpora available, and 2) the feed-back obtained from end-users (marketing experts and trained terminologists) by interviewing and observation-based methods.

5.1 Experimental Results

Two data sets of highly specialised marketing documents (i.e. expert-to-expert marketing documents) were considered: a) a training and validation set **TV** which contains a selection of 72 documents that attempt to be an overall overview of the field of marketing and which includes case studies, chapters from books, journal papers, etc, and b) a test set **TE** which contains 60 marketing articles, and its scope is wide enough to cover the spectrum of marketing's sub-disciplines.

In table 1 the marketing categories considered are listed, where each category is associated with a top level marketing concept from a taxonomy of marketing topics. Table 1 shows the number of documents of the test set **TE** that are manually assign to each category by marketing experts. For a given document d_j, the Web services-based annotation application ranks the seven marketing categories according to their estimated appropriateness to the document d_j (see subsection 4.3), and normalised those values by means of JavaScript functions before showing them to the end-user. Table 1 shows the estimated appropriateness assigned by the Web services-based annotation application to those documents that have being manually assigned to each category. Based on the results obtained with the set **TV**, an experimental threshold was defined: only the documents under "*less than 72% of estimated appropriateness*" can be considered as wrongly classified under a certain category by the Web services-based annotation application because the evaluation sessions performed with different marketing experts indicate that a) it is usual to assign more than one category for a given marketing document, and b) marketing experts do not always easily agree about the more appropriate category for a given marketing document.

Table 1. Test set **TE**. Documents assigned to each category manually and automatically.

Categories	**Number of documents manually assigned by marketing experts**	**Web services-based annotation application: estimated appropriateness**		
		100 %	*Equal or more than 72%*	*Less than 72 %*
MC_Customer_ and_Marketing	**8**	3	2	3
MC_Market_ Segmentation	**5**	4		1
MC_Product_and _ Services	**15**	15		
MC_Price	**8**	8		
MC_Place	**4**	2	2	
MC_Promotion	**6**	6		
MC_Marketing_ Research	**14**	8	4	2

According to the reasons above-mentioned, only 10% of the marketing documents of the test set **TE** can be considered as wrongly classified. This result combined with the relatively high level of estimated appropriateness (*equal or more than 72%*) encourage the current approach to extend the study among the vast marketing corpora available to verify if the current approach facilitates the economical annotation of large document collections.

In order to improve the categorisation capability of the Web services-based annotation application, it may be needed to check the correctness of the selected categories and modify or insert new terms into the thesaurus. To illustrate this: among the finite set of marketing categories considered in table 1, the category that has the biggest categorisation error (37.5 %) is *MC_Customer_ and_Marketing*. With the aim of bringing some light about the causes of the relatively high categorisation error found for the category *MC_Customer_ and_Mark*, different marketing experts were consulted. The consultation revealed that marketing experts strongly disagree about the selection of terminological entries from a thesaurus and their associations with the category *MC_Customer_ and_Marketing*, and therefore, a refinement in the semantics (meaning) of the category and in the terminological entries associated are needed.

5.2 End-Users Feed-Back

Several evaluation sessions were performed to obtain feed-back from marketing experts and trained terminologists. During those sessions, the Think-Aloud Protocol (TAP) [43] was frequently used to gain an outline about the efficacy of the Web services-based annotation application. TAP is a verbal protocol method popularly used to gather usability data during system evaluation by asking the users to vocalize their thoughts, feelings and opinions concurrently while interacting with the system. The audio recorded data reveals that comments like "*incredible quick*", "*just at the click of a button*", or "*is quite right*" appear frequently. These comments enlightened the fact that the Web services-based annotation application has reduced substantially the time that marketing experts and trained terminologists have to invest to classify a highly specialised document about marketing (i.e. expert-to-expert marketing documents) and extract/highlight a minimum of relevant marketing terminology from the document from an average of hours to an average of seconds. Furthermore, the above-mentioned tasks have been simplified to just a click of a button.

6 Conclusions

The approach highlights the benefits of Semantic Web technologies in order to obtain a combination of Web services that easily deal with terminological background knowledge to automate the semantic annotation and text categorisation of highly specialised documents (i.e. expert-to-expert documents). On the one hand, although OWL-S [12] is not currently ready to support the dynamic discovery, composition, and invocation of services; OWL-S facilitates to define the inputs and outputs of a service in terms of an ontology which is a step forward to enable dynamic discovery, composition, and invocation of services without user intervention. On the other hand, the OWL Web Ontology Language XML Presentation Syntax [14] has been exposed as a good way of encoding OWL [11] domain ontology fragments as XML documents that are fruitful to be passed between Web services and that may be needed by other components in the same workflow.

The substantial reduction in the time and effort required from marketing experts and trained terminologists to classify highly specialised marketing documents (i.e. expert-to-expert marketing documents) and extract/ highlight a minimum of relevant marketing terminology from documents, together with a high accuracy in the automatic text categorisation performed by the Web services-based annotation application, encourage the current research study to be extended to a large collection of documents. Furthermore, from the point of view of terminologists, translators and interpreters as well as translator and interpreter trainers it is a paramount to have an easy-to-use environment or tool that not only provides terms already available from a controlled vocabulary or thesaurus, but more importantly shows/highlights very quick those terms in context (how terms are being used in highly specialised documents).

References

1. Amardeilh, F., Laublet, P., Minel, J.L.: Document annotation and ontology population from linguistic extractions. In: Proceedings of the 3rd international conference on Knowledge Capture, pp. 161–168 (2005)
2. Corcho, O.: Ontology based document annotation: trends and open research problems. International Journal of Metadata, Semantics and Ontologies 1, 47–57 (2006)
3. Kahan, J., Koivunen, M.R., Prud'Hommeaux, E., Swick, R.R.: Annotea: An Open RDF Infrastructure for Shared Web Annotations. In: Proceedings of the 10th International World Wide Web Conference (WWW10), Hong Kong, China (2001)
4. Kettler, B., Starz, J., Miller, W., Haglich, P.: A Template-based Markup Tool for Semantic Web Content. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, Springer, Heidelberg (2005)
5. OntoMat-Annotizer, http://annotation.semanticweb.org/ontomat/index.html
6. Kalyanpur, A., Hendler, J., Parsia, B., Golbeck, J.: SMORE – Semantic Markup. In: RDF (ed.) Ontology (2005), Available at: http://www.mindswap.org/papers/SMORE.pdf
7. The SHOE Knowledge Annotator, http://www.cs.umd.edu/projects/plus/SHOE/ KnowledgeAnnotator.html
8. Benjamins, V.R., Contreras, J., Blázquez, M., Dodero, J.M., García, A., Navas, E., Hernández, F., Wert, C.: Cultural heritage and the semantic web. In: Bussler, C., Davies, J., Fensel, D., Studer, R. (eds.) The Semantic Web: Research and Applications, First European Semantic Web Symposium, pp. 433–444. Springer-Verlag, Heidelberg (2004)
9. Saha, http://www.seco.tkk.fi/applications/saha/
10. Medjahed, B., Bouguettaya, A.: A multilevel composability model for semantic Web services. IEEE Transactions on Knowledge and Data Engineering 17(7), 954–968 (2005)
11. OWL, http://www.w3.org/2004/OWL/
12. David, L.: Bringing Semantics to Web Services: The OWL-S Approach. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, Springer, Heidelberg (2005)
13. Tsur, S., Abiteboul, S., Agrawal, R., Dayal, U., Klein, J., Weikum, G.: Are Web Services the Next Revolution in e-Commerce (Panel). In: Proc. Very Large Data Bases Conf., pp. 614–617 (2001)
14. Hori, M., Euzenat, J., Patel-Schneider, P.F.: OWL web ontology language XML presentation syntax. W3C Note (2003), Available at http://www.w3.org/TR/owl-xmlsyntax/
15. Dublin Core, http://dublincore.org/documents/dces/
16. Medical Subject Headings (MeSH), http://www.nlm.nih.gov/mesh/meshhome.html
17. SWRC ontology, http://ontoware.org/projects/swrc/
18. Benjamins, V.R., Fensel, D.: Community is knowledge (KA)2. In: Proceedings of the 11th Workshop on Knowledge Acquisition, Modeling, and Management, Banff, Canada (1998)
19. OntoWeb, http://www.ontoweb.org
20. Hyvönen, E.: Semantic Web Applications in the Public Sector in Finland - Building the Basis for a National Semantic Web Infrastructure. Norwegian Semantic Days, Stavanger, Norway (2006)

21. Gruber, T.R.: A translation approach to portable ontologies. Knowledge Acquisition 5(2), 199–220 (1993)
22. McIlraith, S., Song, T., Zeng, H.: Semantic Web services. IEEE Intelligent Systems, Special Issue on the Semantic Web 16, 46–53 (2001)
23. Uren, V., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E., Ciravegna, F.: Semantic Annotation for Knowledge Management: Requirements and a survey of the state of the art. Journal of Web Semantics 4(1) (2006)
24. Ajax, http://adaptivepath.com/publications/essays/archives/000385.php
25. http://www.w3.org/TR/owl-xmlsyntax/owlxml2rdf.xsl
26. Protégé, http://protege.stanford.edu/
27. Elenius, D., Denker, G., Martin, D., Gilham, F., Khouri, J., Sadaati, S., Senanayake, R.: The OWL-S. In: Gomez-Perez, Euzenat (eds.) A Development Tool for Semantic Web Services, pp. 78–92 (2005)
28. WSDL, http://www.w3.org/TR/wsdl20
29. SOAP, http://www.w3.org/TR/soap12-part0/
30. WS-BPEL, http://www.ibm.com/developerworks/library/specification/ws-bpel/
31. WSMO working group. D16.1v0.2 The Web Service Modeling Language WSML, WSML
32. Davies, J., Fensel, D., van Harmelen, F. (eds.): Towards the Semantic Web: Ontology-Driven Knowledge Management. John Wiley, Chichester (2002)
33. Schreiber, A., Akkermans, H., Anjewierden, A.A., Hoog, R., Shadbolt, N.R., Van de Velde, W., Wielinga, B.: Engineering and managing knowledge. In: The CommonKADS methodology, The MIT Press, Cambridge (1999)
34. Fernandez Prieto, M.J., Moroto Garcia, N.: The SALCA project: marketing terminology in Spanish and English. In: Thelen, M., Lewandowska-Tomasczyk (eds.) Translation and Meaning Part 5, Maastricht:Hogeschool Zuyd, Maastricht School of Translation and Interpreting, pp. 231–239 (2001)
35. Dictionary of Marketing Terms, http://www.marketingpower.com/mg-dictionary.php
36. Monash University: marketing dictionary, http://www.buseco.monash.edu.au/mkt/ dictionary/
37. Erdmann, M., Studer, R.: Ontologies as conceptual models for XML documents. In: Proceedings of KAW 1999, Banff, Canada (1999)
38. XML Schema, http://www.w3.org/XML/Schema
39. TERMIUM, http://www.termium.gc.ca/
40. EURODICAUTOM, http://iate.europa.eu/
41. Sebastiani, F.: Machine Learning in Automated Text Categorization. ACM Computing Surveys 34(1), 1–47 (2002)
42. Lewis, D.D.: Naive (Bayes) at forty: The independence assumption in information retrieval. In: Nédellec, C., Rouveirol, C. (eds.) Machine Learning: ECML-98. LNCS, vol. 1398, pp. 4–15. Springer, Heidelberg (1998)
43. Ericsson, K.A., Simon, H.A.: Protocol analysis: verbal reports as data. MIT Press, Cambridge (1984)

Ontology Based Categorization in eGovernment Application

Claude Moulin[1], Fathia Bettahar[1], Jean-Paul Barthès[1], and Marco Luca Sbodio[2]

[1] University of Compiègne, CNRS, Heudiasyc, Centre de Recherches de Royallieu, 60205 Compiègne, France
[2] Italy Innovation Center, Hewlett Packard Italiana, C.so Trapani 16, 10139 Torino, Italy

Abstract. Applications in eGovernment domain often manage a knowledge base containing information about citizens. In this domain many commissions have to statute on citizens conditions in order to allow some assistance. Processes for classifying citizens have to be proposed to simplify the work of these commissions.

We propose a method for automatically classify instances of concepts in knowledge bases. In this categorization, instances may themselves belong to a category defined by a rule or may be associated to specific instances or concepts defined in an ontology. We consider three types of classification that can be applied to the main criteria of social care applications.

We also present a module allowing tools to get all required information about the categorization of elements in a knowledge base and in particular the categorization of citizens.

1 Introduction

Applications in eGovernment domain often manage a knowledge base containing information about citizens. It is interesting to query such a knowledge base in order to select elements having similar properties. These information often concern persons but all the considerations concerning the categorization could be applied to any kind of elements. For example, it is valuable to know people that can benefit of some social services with regard to their situation. The characteristics that have to be taken into account for the categorization may concern in particular the age, the conditions of unemployment, the handicap, the health of a person.

When defining the notions that allow to classify the persons according to some conditions, we use both assertions of the ontologies, and knowledge expressed by rules. These notions allows to categorize persons of a knowledge base seen as individuals of `Person` concept of an ontology.

The categorization depends on modeling of the ontology. However, we took into consideration different ways of defining some pertinent ontological elements. We propose a solution that can be applied locally according to the way the ontologies are modeled and used in actual applications. The analysis of situations lead to consider three ways of categorization of knowledge base elements. These ways are illustrated by simple examples. However, in real applications developed during the TERREGOV project (see section 4), the modeling is more complicated. We present each case theoretically.

We also present the Java library that we have built, and which allows to query an RDF knowledge base in order to have a better representation of its content. It mainly uses the Jena framework [1]. This library takes as input all OWL files containing the ontologies referenced by the knowledge base and their extensions, the OWL files (or the Jena models) representing the knowledge base to analyze, and the rule file describing the conditions used for classifying the elements. We detail how to integrate the required elements allowing the library to give the categorization results.

In this paper, we consider that the schema of the RDF knowledge base is given by one or more ontologies that can be locally extended. Ontologies and knowledge bases have an OWL representation. The vocabulary used in explanations is the one generally associated with the OWL paradigm.

We also consider that the possible users are in the following situation: they manage a knowledge base whose elements are associated with some ontologies, but they are not the authors of these ontologies and they can not modify them. They wish to know how to use this tool and what are the files to create or adapt. In particular, when some ontological extensions have to be made, they will be created with a namespace different from the namespaces of the ontologies used in the knowledge base. In this paper, all ontology examples have been inserted using the N3 format.

2 Three Types of Categorization

2.1 Ontology

The solution we proposed in the TERREGOV project is based on the definition of an ontology of the social care domain. The ontology is designed to be inserted and used by different modules of a platform allowing civil servants to deliver services to citizens. The ontology was built by several groups of experts following a classic methodology (see [1] for more details). We applied in this sense the principle that terms must be accessible to future users [2,3,4]. We added a mechanism allowing to build indexes from all important information found in the ontology [5].

An ontology representation structure may be very expressive, but that may lead to difficulties when reasoning. Conversely, a restriction of the knowledge representation allows better control of the reasoning task, but may prevent building some concepts [6]. It is thus necessary to follow a method which preserves the expressive nature of the ontology and insures the control of the reasoning task. We chose to extend the ontology with rules [7] like those presented in the following sections in order to create new assertions in knowledge bases.

[1] See http://jena.sourceforge.net

Rules allow engines to create the assertions that classify instances in knowledge bases without modifying the structure of the ontology. This is very important in the e-Government domain when laws change [8].

2.2 Virtual Class

Let's suppose that we want to find the elderly people described in a knowledge base. Let's suppose that the ontology contains the concept of `<Person>`, domain of the datatype property `<age>`. We are interesting in knowing the instances of `<Person>` of the knowledge base whose age is greater or equal to 65. This rule gives us the definition of the category of elderly people. The problem is to represent this category in the knowledge base.

We chose to add a class, subclass of `<Person>`, called `<ElderlyPeople>`. However, it is not possible to directly instantiate this class: only an authorized module like a knowledge base engine can do that. For that reason, we called this kind of class a virtual class. The fact that an instance of `<Person>` belongs to `<ElderlyPeople>` or not depends on a condition (rule) that has to be verified only when the request is made.

However, as any other concept this class must be defined clearly and without ambiguity. It must at least have labels and comments about the way to consider it. As said in the introduction we prefer to add this information in an ontology that extends the ontology where `<Person>` is defined. An ontology designer could also introduce virtual classes in the original ontology. The tool we have built (see section 3.1) take this eventuality in consideration. The `<ElderlyPeople>` class must also be declared as a subclass of `<Person>`. It is necessary for an engine to distinguish a normal class and a virtual class. For that reason we declare a specific class, the class of all the virtual classes, called `<VirtualClass>`. In the following example `tg` indicates the namespace of the main ontology.

```
:VirtualClass
      a       rdfs:Class ;
      rdfs:comment "Class of all the virtual classes"@en ;
      rdfs:label "Virtual Class"@en ;
      rdfs:subClassOf owl:Class .

tgkb:ElderlyPeople
      a       :VirtualClass , owl:Class ;
      rdfs:comment "Persons that are more than 65"@en ;
      rdfs:label "Elderly People"@en ;
      rdfs:subClassOf tg:Person .
```

An instance of `<Person>` belongs to the `<Elderly_People>` class is defined by a condition given by a rule: if a `<Person>` instance has an age greater than 65, then it is an instance of `<Elderly_People>`. It is only a sufficient condition for an instance to belong to this new class. However it is the only way for a instance to belong to the `<ElderlyPeople>`. Rule are written in the Jena format. The namespace of the knowledge base is represented by `tgkb`.

```
[ Role_Elderly:
  (?p rdf:type tgkb:Elderly_People)
  <-
  (?p rdf:type tg:Person)
  (?p tgkb:hasAge ?y)
  ge(?y, 65)  // greater or equals to
]
```

Virtual classes may require the definition of datatype properties whose values are calculated from information occurring in the knowledge base. It is the case of the `<Elderly_People>` class. The previous example was a little bit simplified in order to introduce the concept of virtual class. The age of a person is not directly inserted in the knowledge base because it changes every year. The property age is not an attribute of the `<Person>` class. The actual attribute is `<date_of_birth>`. We have created a new type of data properties that does not lead to a category of elements, but which is useful in the description of knowledge. We call Virtual Datatype Properties, those datatype properties whose definition depends on other properties to be calculated. They can not be directly instantiated. The `<VirtualDatatypeProperty>` class allows an engine to distinguish a normal and a virtual datatype property. In the following example the age is first declared as a virtual data type property and the age of a person is then defined by a rule:

```
:hasAge
      a       :VirtualDatatypeProperty ;
      rdfs:comment "gives the age of a person"@en ;
      rdfs:domain tg:Person ;
      rdfs:label "has age"@en ;
      rdfs:range xsd:int .

[ Role_hasAge:
(?p tgkb:hasAge ?y)
    <-
    (?p rdf:type tg:Person)
    (?p tg:hasBirthDate ?d)
     getYears(?d, ?y)
]
```

Notice that in this definition we use the possibility offered by the rule format to create custom operators. In the last line `getYears` is an operator that calculates the age of a person by comparing the current date to the birth date contained in the `?d` variable and affects the result to the `?y` variable. The elderly rule seen above allows the engine to create assertions in the knowledge base from instances that can be associated to the virtual class. In the following example, the Lana description leads to assign Lana to elderly people:

```
:lana
a  tg:DisabledPerson ;
   tg:hasBirthDate "1940-01-18"^^xsd:date ;
   ...
   tg:hasName "Lana"^^xsd:Name .
```

2.3 Virtual Object Property

Let's suppose that we want to know the type of help a citizen may receive. A person may benefit from some housing or financial help according to some conditions, based on age, health, revenue or unemployment situations. Let's suppose that the ontology contains the concept of `<Help_Types>`, and some instances of it, such as `<housing_help>`, `<financial_help>`, `<medical_help>`. Instances of this class represent the types of help a citizen may benefit from. We associate a person with the type of help she can benefit from, by building a new property linking the `<Person>` class and the `<Help_Types>` class. Thus, the second way to categorize people is to use an object property. This relation can only be instantiated by knowledge base engines and is still called virtual and declared as a virtual object property.

```
:hasHelp
  a :VirtualObjectProperty ;
  rdfs:comment "indicates the type of help
      a person can benefit from"@en ;
  rdfs:domain tg:Person ;
  rdfs:label "has help"@en ;
  rdfs:range tg:Help_Types .
```

It is necessary for an engine to distinguish a normal object property and a virtual object property. For that reason we declare a specific class, the class of all the virtual object properties, called `VirtualObjectProperty`. The conditions for a person to benefit of help depend on information occurring in the knowledge base that uses concepts and properties of the reference ontology. These conditions are expressed with rules. We give a simplified rule that illustrates a sufficient condition.

```
[ Role_hasMedicalHelp:
  (?p tgkb:hasHelp tgkb:medicalHelp)
  <-
  (?p rdf:type tg:DisabledPerson)
  (?p rdf:type tgkb:Elderly_People)
]
```

The rule shows that a disabled and elderly person can benefit of the medical help. Obviously other rules could assign the same type of help to a person but under other conditions and other rules assign other types of help. Rules can embed virtual datatype properties and virtual classes. The engine will create assertions in the knowledge base for instances that satisfy the conditions.

```
:lana
   a  tg:DisabledPerson, tgkb:Elderly_People ;
   tg:hasBirthDate "1940-01-18"^^xsd:date ;
   tgkb:hasHelp tgkb:medicalHelp ;
   tg:hasName "Lana"^^xsd:Name .
```

2.4 Virtual Potential Property

Let's suppose that we want to know what kind of services a person may benefit of. For example, a person may benefit of an housing service, a kind of social service according to some conditions, based on handicap. Let's suppose that the services like <Housing_Service> are modeled as concepts, subclasses of the <Social_Service> concept.

An instance of such a service can be seen as a sort of contract between an administration and a person. When the case of a person is studied, the contract does not exist yet and thus no instance of the service occurs in the knowledge base.

A property <has_housing_service> can be defined for the <Person> concept. The problem is to define the instances of such a property. Basically we could associate to each <Person> fulfilling the conditions the <Housing_Service> concept itself. However, in this case the range of such a property should be owl:Class, which is not compliant with the OWL-DL level. The solution is to set the range of the <has_housing_service> property to <Housing_Service>, and to create an anonymous instance of <Housing_Service> when the property <has_housing_service> is examined.

We qualify such a property a Virtual Potential Property, virtual for the same previous reason and Potential because the instance of the property range to link to the instance of <Person> does not exist when the property is defined.

```
:hasHousingService
  a  :VirtualPotentialObjectProperty ;
  rdfs:domain tg:Z-Person ;
  rdfs:range tg:HousingService .
```

It is necessary for an engine to distinguish a virtual object property and a virtual potential object property. For that reason we declare a specific class called VirtualPotentialObjectProperty, the class of all virtual potential object properties .

The conditions for a person to benefit of such a service depends on information contained in the knowledge base that uses concepts and properties of the reference ontology. These conditions are expressed with rules. We give a simple rule that illustrates a sufficient condition. In this example, a ?p variable is associated to an ?x variable if ?p satisfies the condition. The last line of the rule states that a blank node is created in the knowledge base; such blank node is an instance of <Housing_Service> and associated to elements binded to ?p.

```
[ Rule_hasHousingService:
  (?p, tgkb:hasHousingService, ?x)
  <-
  (?p rdf:type tg:Z-DisabledPerson)
  makeTemp(?x)
]
```

3 Tool

3.1 Technology

We have chosen an homogeneous suite of technologies in order to implement a tool able to answer the different cases of categorization we have presented in the previous sections. The tool helps the user to query a RDF knowledge base for categorization purpose. The ontologies that the tool imports must be OWL-DL compliant. The tool is written in Java (at least version 5.0). Ontologies and knowledge bases are parsed using the Jena technology. Queries against knowledge base are expressed in SPARQL. Rules respect the Jena rule syntax.

The tool takes in input different elements describing all the required knowledge. They consist in one or more ontologies, the knowledge base and the rule description. One of the ontology defines the virtual classes respecting the identifier syntax presented in the examples. It may happen to import several ontologies declaring these classes. Each virtual concept or property must be declared as an instance of one of these classes.

This rule model allows to create the anonymous instances allowing the third case of categorization (through virtual potential object properties). The other advantage of this model is the possibility to define custom operators (like the GetYears in the example) which can be added to the system. We have extended the parsing of the rule file in order to declare custom operators at the beginning of the rule file, adding lines such as:.

```
####### Operator List
# operator=net.eupm.terregov.categorization.jena.GetYears
...
```

3.2 Main Methods

Our tool contains different methods that query the knowledge base. Their main role is to encapsulate the SPARQL queries against the knowledge base and the corresponding result sets. It is possible for a client application to directly query the knowledge base with the appropriate method. The elements bound to the query variables are returned in an array. The main methods of this library return: the instances of a virtual class, the instances of a virtual object property (a list of couples containing an instance of the property domain and an instance of the property range), the instances of a virtual potential object property (it is a list of couples containing an instance of the property domain and the concept having an anonymous instance associated to it).

As explained in the last example, we can look for the persons that can benefit from a service. These services are modeled as subclasses of a `<Social_Service>`. For that it is necessary to create a virtual potential object property for each service a person may benefit of which may be painful. We have simplified the discovery with the declaration of a generic property, say `<has_Service>` whose domain and range are `<Person>` and `<Social_Service>`. The virtual potential object properties are sub properties of this last one. A method taking in input this generic property returns a list of triples containing: an instance of the generic property domain, a potential property and the range of this potential property.

4 Conclusion

In this paper we have presented three cases of the categorization of knowledge base elements. They are covering all the cases occurring in the scenarios developed by the pilot partners of the TERREGOV project. Our considerations are generic enough to be applied in other domains. The first case of categorization corresponds to the creation of a new class of the ontology. The other cases depend on the way the reference ontology is built. Ontology designers may represent knowledge using classes or individuals and our model supports both.

The application of rules adds permanent assertions, because the status of a person generally remains valid when it is reached (adult or an elderly people). In other case, it is possible to add assertions that are valid only during the queries.

Acknowledgments

This work has been developed within the TERREGOV project, an integrated project co-funded by the European Commission[2] under the IST Program, e-Government unit, under the reference IST-2002-507749.

References

1. Fernandez-Lopez, M.: Overview of methodologies for building ontologies. In: IJCAI 1999. Workshop on Ontologies and Problem-Solving Methods, Stockholm (1999) 4/1,4/13
2. Fikes, R., Farquhar, A.: Distributed repositories of highly expressive reusable ontologies. IEEE Intelligent Systems, 73–79 (1999)
3. Uschold, M., King, M.: Towards a methodology for building ontologies. In: Workshop on Basic Ontological Issues in Knowledge Sharing (1995)
4. Staab, S., Studer, R., Schnurr, H.P., Sure, Y.: Knowledge process and ontologies. IEEE Intelligent Systems, 26–34 (2001)
5. Bettahar, F., Moulin, C., Barthès, J.P.: Adding an index mechanism to an ontology. In: AWIC 2007. 5th Atlantic Web Intelligence, Fontainebleau, France, pp. 56–61 (2007)
6. Doyle, J., Patil, R.: Two dogmas of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. In: rapport, MIT, Cambridge (1989)
7. Golbreich, C., Bierlaire, O., Dameron, O., Gibaud, B.: Use case: Ontology with rules for identifying brain anatomical structures. In: W3C Workshop on Rule Languages for Interoperability, Washington, USA (2005)
8. Ae Chun, S., Atluri, V.: Ontology-based workflow change management for flexible egovernment service delivery. In: The National Conference on Digital Government Research, Boston, USA (2003)

[2] The content of this paper is the sole responsibility of the authors and in no way represents the views of the European Commission or its services.

Semantic Matching Based on Enterprise Ontologies

Andreas Billig, Eva Blomqvist, and Feiyu Lin

Jönköping University, Jönköping, Sweden
{bill,blev,life}@jth.hj.se

Abstract. Semantic Web technologies have in recent years started to also find their way into the world of commercial enterprises. Enterprise ontologies can be used as a basis for determining the relevance of information with respect to the enterprise. The interests of individuals can be expressed by means of the enterprise ontology. The main contribution of our approach is the integration of point set distance measures with a modified semantic distance measure for pair-wise concept distance calculation. Our combined measure can be used to determine the intra-ontological distance between sub-ontologies.

1 Introduction

Our approach presented in this paper is a step towards reducing the information overload and assisting the human user in processing and evaluating the information entering a company from the surrounding world. An enterprise ontology represents, among other things, the business interests and processes of the company, and the information demand of individual users can be expressed in terms of this enterprise ontology. The core algorithm needed for evaluating how distant (or similar) two parts of the enterprise ontology are is the main focus of this paper, in order to support a relevance evaluation of incoming information.

The following section presents background and motivation. In Section 3 the problem is outlined and our semantic distance algorithm is described in detail. Section 4 focuses on a small example to illustrate the usefulness of the method. Finally, in Section 5 some conclusions are drawn and future work is outlined.

2 Background

This section describes the motivation and background of our approach, for example the notion of semantic distance. With respect to ontologies we adopt the classic definition from [1], describing an ontology as a formal explicit specification of a shared conceptualisation. In our research we do not, at this time, restrict ourselves to a specific ontology formalism, but assume the possibility to reduce the ontology to a semantic net-like structure (i.e. a labelled directed graph). Our research focuses mainly on domain and application ontologies within enterprises, i.e. an ontology describing the necessary views and concepts, with the intention of structuring and retrieving information.

The term semantic distance is in this paper used as the inverse of semantic similarity. For semantic distance (or similarity) of ontologies many measures exist to measure distance of concept pairs within one ontology (see the survey in [3]). As described in [6] there are edge-based, information content-based, and feature-based approaches. We want to rely only on the ontology without incorporating documents for distance measurement, thus we do not consider information content approaches. Feature-based methods focus on property definitions. Our approach is based on semantic network-like representations where taxonomies play a more crucial role than property definitions. Still we incorporate user defined relations as well, but we focus on edge-based methods, as the one in [5].

Different aggregation schemes can then be applied to determine distance between sets of objects. Most of these assume a way to determine the distance of two individual objects, from the two sets. In [7] different distance measures are discussed and compared, as well as in [8]. There exist measures such as the simple Hausdorff metric, considering only the most extreme points in the set, and others such as the family of optimal mappings, also incorporating the cardinalities of the sets.

Document relevance in classical IR is commonly measured with respect to some query (a set of keywords), and the document is treated as a bag-of-words [9]. Some systems also use ontologies, as in [10] where RDF annotations are used to represent the meaning of documents. Also in [11] documents are ranked according to an ontology. However, the annotations for the documents are manually defined, in our case we need to avoid manual annotation. Therefore we choose to align the representation of the document to the ontology automatically, using simple string matching (see [2]).

3 The Semantic Matching Approach

This section describes our approach to semantic matching, calculating the semantic distance between two parts of an ontology.

3.1 General Framework

First of all, we rely on an enterprise ontology (see Figure 1). Secondly, we assume that for all users a profile, a set of concepts of the enterprise ontology, exists. As documents, the approach may deal with any kind of machine processable information that can be mapped to a list of terms. The matched part of the term list together with the enterprise ontology form the extended enterprise ontology (XO, as described later in Section 3.3).

To calculate the relevance of a document to a user's interests the following process steps are needed, (1) processing of the document to construct a corresponding term list, (2) aligning the term list to the enterprise ontology (constructing the XO), and (3) calculating the semantic distance.

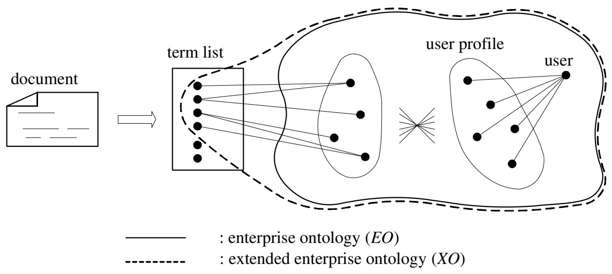

document
term list
user profile
user
: enterprise ontology (*EO*)
: extended enterprise ontology (*XO*)

Fig. 1. Relevant information objects

3.2 Preliminaries

Representing an ontology as a triple set is adequate for our purposes and aligned to RDF [12]. Because there is no need to consider literals here, we restrict ourselves to the set *Res* of (RDF-)resources (subsequently called *concepts*).

An *ontology* is a set of triples (s, p, o) over *Res* where s, p, o is called *subject*, *property*, and *object*, respectively. Let O be an ontology, then *root* denotes the taxonomic root, *Prop* denotes the set of properties occurring in O and $propCard(s,p) = \big|\{o \mid (s,p,o) \in O\}\big|$. An *(ontology) path of length n between concepts x and y* is a sequence of triples $(x, p_1, o_1), \ldots, (s_n, p_n, y)$ where $o_i = s_{i+1}$ for $i = 1, \ldots, n-1$. If triple t is within a path p we also write $t \in p$. *Path* resp. $Path(x,y)$ denote the *set of all paths* resp. the *set of paths between x and y*. A *triple weight* is a function from O to $\mathbb{R}$. A *path weight* is a function from *Path* to $\mathbb{R}$. For a triple weight f and path x the *canonical path weight* κ_f is defined by $\kappa_f(x) = \sum_{i=1,\ldots,n} f(x_i)$ where $x = x_1, \ldots, x_n$. The normalisation of the canonical path weight is defined by $\kappa_f^*(x) = \frac{\kappa_f(x)}{m}$ where $m = max_{y \in Path}\ \kappa_f(y)$. Note that κ_f^* is normalised if f is normalised. The *minimal path weight between x and y according to path weight f restricted by property set R* is defined as: $MinPathWeight(x,y,f,R) = min_{p\in P}\ f(p)$ where $P = \{p \in Path(x,y) \mid \forall (u,v,w) \in p\ .\ v \in R\}$.[1] A *property weight range* is a function $f : Prop \longrightarrow \mathbb{R} \times \mathbb{R}$ where $r_1 \leq r_2$ holds for every $(r_1, r_2) \in f(Prop)$.

3.3 Process Steps

The first step of the process is generating the term list, which is done through standard text processing methods (mainly tokenisation, stop-word removal and stemming, see [9]). The following steps of the process are described below.

Term List Alignment. The term list is related to the enterprise ontology through string matching (see [2]). For this purposes we choose a threshold `t` for the string similarity level. Assuming that $L = TL(d)$ is the term list of

[1] If there is no property restriction, i.e. $R = Prop$, we write $MinPathWeight(x, y, f)$.

processing document d, the *string match of* L is defined by $SM(L) = \{x \in L \mid \exists y \in EO \;.\; \sigma(x, y) \geq \mathsf{t}\}$, where σ is any string matching method that deliverers values between 0 and 1. Then the XO is constructed with the matched terms of the term list as concepts and σ as an ontological relation. The result is the following ontology, where sm stands for the string match relationship:

$$XO = EO \cup \{(x, \mathsf{sm}, y) \mid x \in SM(L),\; y \in EO\}$$

Semantic Distance. Next, is the calculation of the semantic distance between $SM(L)$ and the users' interest profiles. The interest profile of user u is a set of concepts, i.e. $UP(u) = \{r \mid (u, \mathit{interestedIn}, r) \in EO\}$.

Among the edge-based methods the one proposed by Sussna in [5] includes the facility to incorporate user defined relations as well as taxonomic relations, and additionally allows for weighting of relations. Our adaptation of this to the RDF environment is the triple weight ω:

$$\omega(x, p, y) = \frac{w(x, p) + w(y, p)}{2 * \mathit{max}(l(x), l(y))} \;,$$

$$w(x, p) = \mathit{max}_p - \frac{\mathit{max}_p - \mathit{min}_p}{\mathit{propCard}(x, p)} \;, \text{ and}$$

$$l(x) = \mathit{length}\big(\mathit{MinPathWeight}(x, \mathit{root}, f, \mathsf{rdfs{:}subClassOf})\big) \;,$$

where x, p, y are resources, $(\mathit{min}_p, \mathit{max}_p)$ is a property weight range, and path weight f is equal to the path length. According to [5] the property weight range of rdf:label should be set to $(0, 0)$ and the ranges of rdfs:subClassOf and rdf:type should be $(1, 2)$. The property weight range of user defined relations can be set to the same range, or a value range can be determined experimentally.

To use this together with the string matching score in XO we define:

$$\Delta(x, p, y) = \begin{cases} \dfrac{\omega(x, p, y)}{\mathit{max}(\omega(EO))}, & \text{if } (x, p, y) \in EO \\ |1 - \sigma(x, y)|, & \text{otherwise,} \end{cases}$$

where σ is a string matching method. Note that Δ is normalised with the maximum value of ω for EO and by definition of σ. Furthermore string similarity σ is inverted to string distance because it has to be aligned to concept distance and concept set distance. Because we are interested in the distance between two arbitrary concepts instead of adjacent concepts we define $\Delta^*(x, y) = \mathit{MinPathWeight}(x, y, \kappa^*_\Delta)$ where x, y are concepts from XO and κ^*_Δ is the normalised canonical path weight.

To calculate the semantic distances of concept sets within ontologies we chose the *minimal linking* approach of [7] which embrace the whole sets (instead of acting locally, like the *Hausdorff* measure). This approach is based on the set of linkings M^l between concept sets A and B. For $r \in M^l$ it holds $\forall a \in A\ \exists b \in B$ with $(a,b) \in r$ and vice versa. For the distance calculation those elements from M^l is chosen for which the sum of concept distances Δ^* is minimal:

$$dist^l(A,B) = min_{r \in M^l}\left(\sum\nolimits_{(a,b)\in r} \frac{\Delta^*(a,b)}{|r|}\right)$$

Finally we introduce a measure for the document d and the user interests u:

$$\Delta^l(d,u) = dist^l(SM(TL(d)), UP(u))$$

4 Document Ranking

Our method could be useful in a range of applications. As a first case we apply it to ranking of incoming e-mails, with respect to user profiles expressed by the enterprise ontology. This case is part of a project called Media Information Logistics (MediaILog), which aims to develop and introduce semantic technologies to improve information supply within companies in the Swedish media industry.

4.1 Ranking Example

In this example the objective is to rank two incoming e-mails (documents) with respect to user profiles in the enterprise ontology. A small part of an enterprise ontology is used for understandability purposes (illustrated in Figure 2). The numbers associated with each relation are the distances between adjacent concepts (user defined relations also have the weight range $(1,2)$). Two profiles are used, profile A and profile B in Figure 2 (in the real world case the language would be Swedish). The document texts can be viewed in Table 1.

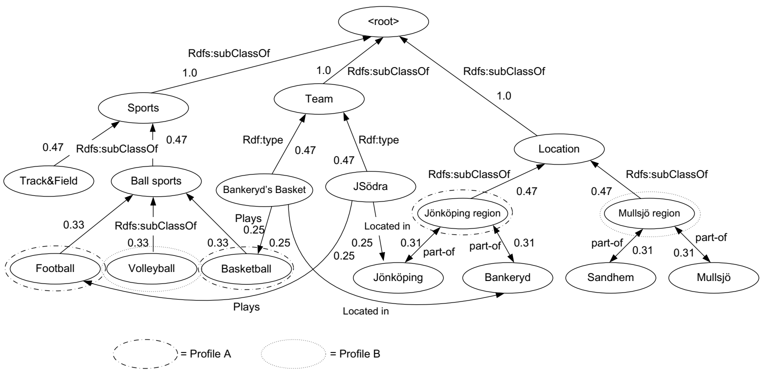


Fig. 2. The example ontology and the two profiles

Table 1. Text content of the documents

Doc#	Text
1	Hi! This weekend JSödra plays a training match against SandhemsIK. It starts on Saturday at 12.00. /Andreas Andersson, JSödra
2	Hi! Bankeryd's Basket invites you to a promotional evening with free snacks and a presentation of our plans for the future of our clubs in the region. The target group is newspapers and local politicians, please spread the word to anyone that might be interested! Regards, Arne Bokvist, Bankeryd's Basket

The term lists are extracted from the texts using standard text processing methods (tokenisation, stop-word removal and stemming, as in [9]). The term lists are matched against the concept labels using string matching (here the Jaro-Winkler measure from [2], with threshold 0.90). The result can be seen in Table 2. So far we do not involve the unmatched concepts in our calculation, but a penalty score could easily be incorporated. The matched terms are temporarily added to the EO to form the XO. An illustration of the XO:s can be seen in Figure 3.

Table 2. The matched terms in the term lists and their string matching scores

Text	Term	Concept	Score	Text	Term	Concept	Score
1	JSödra	JSödra	1.0	2	Bankeryd's	Bankeryd's Basket	0.91
1	SandhemsIK	Sandhem	0.94	2	Basket	Basketball	0.92
2	Bankeryd's	Bankeryd	0.98				

Now, the distances between all concept pairs containing one concept from the extension (representing the text) and one concept in the current profile, can be computed. The values shown in Table 3 result from text 1 and profile A. Finally, these values are aggregated using the point set measure. The resulting ranks are shown in Table 4.

The small differences between the ranking values are due to the very small ontology and due to that the texts are of relatively similar topics. Still it is possible to note benefits of our approach, especially compared to approaches using keywords. Exact matching of keywords would not have been able to determine the

Table 3. Distances between concepts in XO representing profile A and text 1

XO	Profile	Path	Dist.	XO	Profile	Path	Dist.
JSödra	Football	2	0.073	SandhemsIK	Football	7	0.62
JSödra	Basketball	4	0.27	SandhemsIK	Basketball	7	0.62
JSödra	Jönköping region	3	0.19	SandhemsIK	Jönköping region	4	0.38

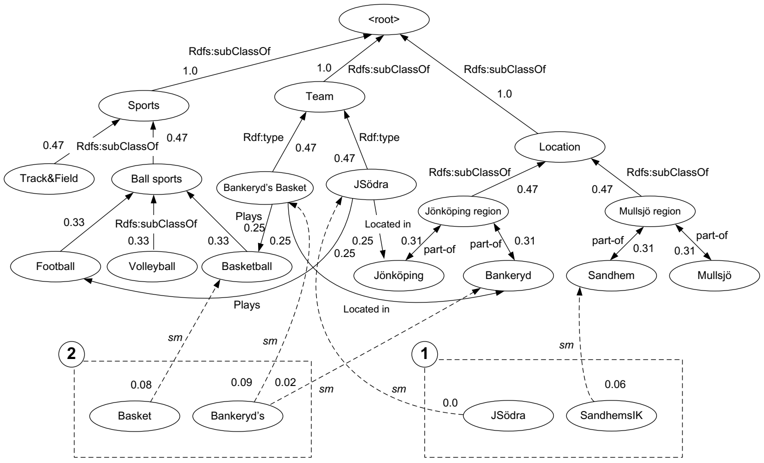


Fig. 3. The extended Enterprise Ontology

relevance of either document, since none of the profile concepts are mentioned in the texts. Inexact string matching would work better but still with no way of determining the relevance of the first document, without using the ontology to discover that for example JSödra is a football team.

Table 4. Ranking of the texts with respects to profiles A and B

	Document 1	Document 2
Profile A	0.76	0.86
Profile B	0.82	0.70

5 Conclusions

In this paper we have presented a general measure for computing the distance between two sets of concepts in the same ontology. The novelty of this approach is mainly that it modifies an existing point set distance measure and then combines it with a modified version of a commonly used distance measure between pairs of concepts in the same ontology. This is a general approach that can be useful in many application scenarios. This is also what we envision in the future of the MediaILog project, for example by assessing news items from news agencies or updates on websites. Intuitively, it is easy to see the basic usefulness of this approach, since it takes into account more background knowledge than a simple keyword-based approach, but still a thorough evaluation is also needed.

Future work concerning the distance measure intend to use a first application to measure and compare its performance to other systems and measures. There may be need for optimisation to reach a suitable time performance, although many parts can also be computed "off-line". We envision the option to use cut-off thresholds for the path computations to reduce the number of possible paths.

Acknowledgements

This work was mainly conducted within the MediaILog research project, financed by the Swedish foundation *Carl-Olof och Jenz Hamrins Stiftelse*. Special thanks to the three anonymous reviewers for valuable comments on how to improve this paper.

References

1. Gruber, T.: A translation approach to portable ontology specifications. Knowledge Acquisition 5, 199–220 (1993)
2. Cohen, W., Ravikumar, P., Fienberg, S.: A Comparison of String Distance Metrics for Name-Matching Tasks. In: Proc. of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03) (August 9-10, 2003), Acapulco, Mexico. (2003)
3. Blanchard, E., Harzallah, M., Briand, H., Kuntz, P.: A typology of ontology-based semantic measures. In: Proc. of the Open Interop Workshop on Enterprise Modelling and Ontologies for Interoperability (2005)
4. Shvaiko, P., Euzenatm, J.: A Survey of Schema-based Matching Approaches. Journal on Data Semantics IV, 146–171 (2005)
5. Sussna, M.: Word sense disambiguation for free-text indexing using a massive semantic network. In: Proceedings of the second international conference on Information and Knowledge Management, ACM Press, New York (1993)
6. Raftopoulou, P., Petrakis, E.: Semantic Similarity Measures: a Comparison Study. Technical report, Technical University of Crete. Department of Electronic and Computer Engineering (2005)
7. Eiter, T., Mannila, H.: Distance measures for point sets and their computation. Acta Informatica (1997)
8. Ramon, J., Bruynooghe, M.: A polynomial time computable metric between point sets. Acta Informatica 37(10) (2001)
9. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley, Reading (1999)
10. Chirita, P.A., Ghita, S., Nejdl, W., Paiu, R.: Semantically enhanced searching and ranking on the desktop. In: Proc. of the ISWC Workshop on The Semantic Desktop Next Generation Personal Information Management and Collaboration Infrastructure, Galway, Ireland (2005)
11. Castells, P., Fernández, M., Vallet, D.: An adaptation of the vector-space model for ontology-based information retrieval. IEEE Transactions on Knowledge and Data Engineering 19, 261–272 (2007)
12. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation. W3C Consortium (2004)

Author Index