

Oscar Pastor  
João Falcão e Cunha (Eds.)

LNCS 3520

# Advanced Information Systems Engineering

17th International Conference, CAiSE 2005  
Porto, Portugal, June 2005  
Proceedings

 Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*University of Dortmund, Germany*

Madhu Sudan

*Massachusetts Institute of Technology, MA, USA*

Demetri Terzopoulos

*New York University, NY, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Moshe Y. Vardi

*Rice University, Houston, TX, USA*

Gerhard Weikum

*Max-Planck Institute of Computer Science, Saarbruecken, Germany*

Oscar Pastor João Falcão e Cunha (Eds.)

# Advanced Information Systems Engineering

17th International Conference, CAiSE 2005  
Porto, Portugal, June 13-17, 2005  
Proceedings

Volume Editors

Oscar Pastor

Valencia University of Technology

Department of Information Systems and Computing

Camí de Vera s/n, 46022 València, Spain

E-mail: opastor@dsic.upv.es

João Falcão e Cunha

University of Porto

Faculty of Engineering

Centres of Competence in Electronic Commerce, Porto, Portugal

E-mail: jfcunha@fe.up.pt

Library of Congress Control Number: 2005926626

CR Subject Classification (1998): H.2, H.3-5, J.1, K.4.3-4, K.6, D.2, I.2.11

ISSN 0302-9743

ISBN-10 3-540-26095-1 Springer Berlin Heidelberg New York

ISBN-13 978-3-540-26095-0 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

[springeronline.com](http://springeronline.com)

© Springer-Verlag Berlin Heidelberg 2005

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper SPIN: 11431855 06/3142 5 4 3 2 1 0



# Preface

To write a preface means that we have reached the end of this long way, and that in some way all the incidences and problems have been overcome. We can now say that it is really a big pleasure for us to welcome all of you to the proceedings of CAiSE 2005 which was held in Porto.

CAiSE 2005 was the seventeenth in the series of International Conferences on Advanced Information Systems. Enforcing its tradition, since the late 1990s the CAiSE conferences have provided a forum for the presentation and exchange of research results and practical experiences within the field of advanced information systems engineering. In 2005, the conference was hosted by the Faculdade de Engenharia da Universidade do Porto, Portugal, going back to southern Europe, in particular to the Iberian peninsula, where CAiSE 1997 was held in Barcelona.

The conference theme of CAiSE 2005 was “Improving Communication and Understanding — Systems and Citizens.” Behind this theme, we find the fact that the Internet has been changing societies and economies, and the way institutions and businesses operate has evolved rapidly. Most citizens are now expected to interact directly with technology-based systems without direct human intermediation, using different interface appliances. Moreover, many information systems interact with each other with limited human supervision. The Internet and its infrastructure play a critical role in this evolution, and it is of the utmost importance that semantic aspects are taken into consideration on a sound basis. Improved communication and understanding between people, the final objective of advanced information systems, requires improved communication and understanding between systems and their developers. Advanced information systems engineering therefore needs to keep improving its methods and technologies in order to support the proper development of our economies and of our societies.

This challenging proposal of CAiSE 2005 attracted scientists and practitioners from all over the world to submit their contributions. Up to 282 submissions were received, out of which the Program Committee selected 36 top-quality papers. This fierce competition produced an acceptance rate of 13%, which explains in itself the hard selection process that we faced. All the submissions were reviewed by at least three members of the CAiSE 2005 Program Committee, composed of well-known, relevant scientists related to the different CAiSE topics. The resulting program reflected the fact that the topic of information systems engineering encompasses human and organizational issues as well as technical issues, at all levels of the software production process. This includes different issues related to conceptual modelling, metamodelling, databases, query processing, process modelling and workflow systems, requirements engineering, model transformations, knowledge management and verification, Web services, Web engineering, software testing and software quality. All of these subject areas were covered as technical sessions of the final program.

Additionally, the authors of 31 papers presenting emerging ideas were invited to participate in the CAiSE Forum. The CAiSE Forum was initiated by the organizers of CAiSE 2003 in Velden as a means of stimulating scientific debate. It aims at introducing an agora for the presentation of fresh ideas, new concepts, and experience reports on application experiences and project reports, as well as demonstration of new and innovative systems, tools and applications. After the successful experiences of Velden and Riga, the Forum initiative was carried through to Porto.

The success of the CAiSE conferences is shown by the large number of co-located events. CAiSE 2005 was accompanied by 11 workshops and 6 tutorials that attracted a large number of participants. In addition, the edition of WER 2005 was co-located with CAiSE 2005. Furthermore, three very relevant keynote speakers provided a great complement for the technical program: Prof. Antoni Olivé (Universitat Politècnica de Catalunya) with the keynote “Conceptual Schema-Centric Development: a Grand Challenge for Information Systems Research,” Prof. Jean Vanderdonckt (Université de Louvaine-la-Neuve) with the keynote “A MDA-Compliant Environment for Developing User Interfaces of Information Systems,” and Prof. Sudha Ram (University of Arizona) with the keynote “Toward Semantic Interoperability of Heterogeneous Biological Data Sources.” We thank them very much for accepting the invitations.

We devote a special thanks to the members of the Program Committee for doing excellent reviewing work. Their dedicated work was instrumental in putting together yet another high-quality CAiSE conference. We wish also to give special thanks to the local organizers at the “Faculdade de Engenharia da Universidade do Porto” and at the “Departament de Sistemes Informàtics i Computació, Universitat Politècnica de València” for their hard work and devotion. Without their efforts, we would have had just nothing. Thank you very much.

Finally, the CAiSE 2005 organizers also thank the main conference sponsors: UPV, Univ. Politècnica de València, Spain; ERCIM, European Research Consortium for Informatics and Mathematics; UP, Univ. do Porto, Portugal; FEUP, Faculdade de Engenharia da Univ. do Porto, Portugal; FCT, Fundação para a Ciência e Tecnologia, Portugal; FLAD, Fundação Luso Americana para o Desenvolvimento, Portugal. Their support made possible the success of the conference.

Thanks for joining us, and we hope that all participants enjoyed CAiSE 2005!

June 2005

Oscar Pastor  
João Falcão e Cunha

# Organization

## Advisory Committee

Janis Bubenko Jr., Royal Institute of Technology, Sweden  
Colette Rolland, Université de Paris 1, Panthéon, Sorbonne, France  
Arne Sølvsberg, Norwegian University of Science and Technology, Norway

## General Chair

João Falcão e Cunha, Univ. do Porto, Portugal

## Program Chair

Oscar Pastor, Univ. Politècnica de València, Spain

## Local Organization

Raul Moreira Vidal, Univ. do Porto, Portugal  
Henriqueta Nóvoa, Univ. do Porto, Portugal  
José Luís Borges, Univ. do Porto, Portugal  
João Pascoal Faria, Univ. do Porto, Portugal  
Ademar Aguiar, Univ. do Porto, Portugal  
Pedro J. Valderas, Univ. Politècnica de València, Spain  
Victoria Torres, Univ. Politècnica de València, Spain  
Javier Muñoz, Univ. Politècnica de València, Spain

## Workshop Chairs

Jaelson Castro, Univ. Federal de Pernambuco, Brazil  
Ernest Teniente, Univ. Politècnica de Catalunya, Spain

## Tutorials and Panel Chairs

Nuno Nunes, Univ. da Madeira, Madeira, Portugal  
Bernhard Thalheim, Kiel University, Kiel, Germany

## Forum Chairs

Orlando Belo, Univ. do Minho, Portugal  
Johan Eder, University of Klagenfurt, Austria

## **Industrial Chairs**

Sjaak Brinkkemper, Utrecht University, The Netherlands  
Rui Melo, ANETIE, Portugal

## **Publicity and Communications Chairs**

Vicente Pelechano, Univ. Politècnica de València, Spain  
Eduarda Mesquita, Impacto Design, Portugal  
Jorge S. Carneiro, SAGE Infologia, Portugal

## **Doctoral Consortium Chair**

Ana Moreira, Univ. Nova de Lisboa, Portugal

## **Web Chairs**

Nuno Ramos, Mercatura, Portugal  
Joan Fons, Univ. Politècnica de València, Spain

## **Sponsoring Organizations**

UPV, Univ. Politècnica de València, Spain  
ERCIM, European Research Consortium for Informatics and Mathematics  
UP, Univ. do Porto, Portugal  
FEUP, Faculdade de Engenharia da Univ. do Porto, Portugal  
FCT, Fundação para a Ciência e Tecnologia, Portugal  
FLAD, Fundação Luso Americana para o Desenvolvimento, Portugal

## Preconference Workshops

**11th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2005)**

Erik Kamsties, Vincenzo Gervasi,  
Pete Sawyer

**2nd INTEROP-EMOI Open Workshop on Enterprise Modelling and Ontologies for Interoperability (EMOI-INTEROP 2005)**

Michele Missikoff, Yves Pigneur

**6th Workshop on Business Process Modelling, Development, and Support: Business Processes and Support Systems: Design for Flexibility (BPMDS 2005)**

Stewart Green, Gil Regev,  
Pnina Soffer, Jelena Zdravkovic

**10th International Workshop on Exploring Modelling Methods in Systems Analysis and Design (EMMSAD 2005)**

Terry Halpin, Keng Siau,  
John Krogstie

**Ubiquitous Mobile Information and Collaboration Systems (UMICS 2005)**

Moira Norrie, Schahram Dustdar

**Semantic Web for Web-Based Learning: Implications in the Area of Information Systems in Education (SW-WL 2005)**

Marie-Noelle Bessagnet, Danièle Herin

**International Workshop on Web-Oriented Software Technologies (IWWOST 2005)**

Daniel Schwabe, Gustavo Rossi,  
Luis Olsina, Vicente Pelechano

**International Workshop on Data Integration and the Semantic Web (DISWeb 2005)**

Zohra Bellahsene, Isabel Cruz,  
Dimitris Plexousakis

**Philosophical Foundations of Information Systems Engineering (PHISE 2005)**

Esperanza Marcos, Roel Wieringa

**Workshop on Adaptive and Self-managed Enterprise Applications (ASMEA 2005)**

Malu Castellanos, Sara Comai

**2nd International Workshop on Data and Information Quality (DIQ 2005)**

Markus Helfert, Cinzia Cappiello,  
Martin J. Eppler

## Program Committee

- Abramowicz, Witold (Poland)  
 Amaral, Luis (Portugal)  
 Aguiar, Ademar (Portugal)  
 Atzeni, Paolo (Italy)  
 Barzdins, Janis (Latvia)  
 Belo, Orlando (Portugal)  
 Bertino, Elisa (USA)  
 Borges, José Luís (Portugal)  
 Bouzeghoub, Mokrane (France)  
 Bresciani, Paolo (Italy)  
 Brinkkemper, Sjaak  
     (The Netherlands)  
 Caplinskas, Albertas (Lithuania)  
 Casamayor, Juan Carlos (Spain)  
 Castano, Silvana (Italy)  
 Castro, Jaelson (Brazil)  
 Celma, Matilde (Spain)  
 Cunha, Joao (Portugal)  
 de Troyer, Olga (Belgium)  
 Delcambre, Lois (USA)  
 Diaz, Oscar (Spain)  
 Dori, Dov (Israel)  
 Dubois, Eric (Luxembourg)  
 Eder, Johann (Austria)  
 Ehrich, Hans-Dieter (Germany)  
 Ermolejev, Vadim (Ukraine)  
 Faria, João Pascoal (Portugal)  
 Fiadeiro, Jose (UK)  
 Fraternali, Piero (Italy)  
 Gaedke, Martin (Germany)  
 García, Franciso (Spain)  
 Genero, Marcela (Spain)  
 Giorgini, Paolo (Italy)  
 Grundspenkis, Janis (Latvia)  
 Guarino, Nicola (Italy)  
 Gustas, Remigijus (Sweden)  
 Haav, Hele-Mai (Estonia)  
 Halpin, Terry (USA)  
 Henderson-Sellers, Brian (Australia)  
 Hernández, Juan (Spain)  
 Heuser, Carlos A. (Brazil)  
 Jarke, Matthias (Germany)  
 Jeffery, Keith (UK)  
 Jeusfeld, Manfred (The Netherlands)  
 Johannesson, Paul (Sweden)  
 Kangassalo, Hannu (Finland)  
 Kappel, Gerti (Austria)  
 Karagiannis, Dimitris (Austria)  
 Katrib, Miguel (Cuba)  
 Kirikova, Marite (Latvia)  
 Koch, Nora (Germany)  
 Krogstie, John (Norway)  
 Leite, Julio (Brazil)  
 Leonard, Michel (Switzerland)  
 Liddle, Steve (USA)  
 Ling, Tok Wang (Singapore)  
 Loucopoulos, Pericles (UK)  
 Lu, Jianguo (Canada)  
 Maiden, Neil (UK)  
 Marcos, Esperanza (Spain)  
 Mayr, Heinrich (Austria)  
 Missikoff, Michele (Italy)  
 Moreira, Ana (Portugal)  
 Norrie, Moira (Switzerland)  
 Nunes, Nuno (Portugal)  
 Olivé, Antoni (Spain)  
 Olsina, Luis (Argentina)  
 Opdahl, Andreas L. (Norway)  
 Palazzo de Oliveira, Jose (Brazil)  
 Pelechano, Vicente (Spain)  
 Pernici, Barbara (Italy)  
 Persson, Anne (Sweden)  
 Pitt, Jeremy (UK)  
 Poels, Geert (Belgium)  
 Pohl, Klaus (Germany)  
 Ralyte, Jolita (Switzerland)  
 Regnell, Bjorn (Sweden)  
 Rolland, Colette (France)  
 Rossi, Gustavo (Argentina)  
 Saeki, Motoshi (Japan)  
 Schewe, Klaus D. (New Zealand)  
 Schwabe, Daniel (Brazil)  
 Shoval, Peretz (Israel)  
 Sindre, Guttorm (Norway)  
 Snoeck, Monique (Belgium)  
 Song, Il-Yeol (USA)

Stirna, Janis (Sweden)  
Sutcliffe, Alistair (UK)  
Teniente, Ernest (Spain)  
Thalheim, Bernard (Germany)  
Tsalgatidou, Aphrodite (Greece)  
Vallecillo, Antonio (Spain)  
Vanderdonckt, Jean (Belgium)  
Vasilecas, Olegas (Lithuania)

Vassiliou, Yannis (Greece)  
Wagner, Gerd (The Netherlands)  
Wand, Yair (Canada)  
Wangler, Benk (Sweden)  
Welzer, Tatjana (Slovenia)  
Wieringa, Roel (The Netherlands)  
Wohed, Petia (Sweden)  
Yu, Eric (Canada)

## Additional Referees

Abrahao, Silvia  
 Acuña, César J.  
 Ahlbrecht, Peter  
 Alenljung, Beatrice  
 Araújo, João  
 Athanasopoulos, George  
 Backlund, Alexander  
 Backlund, Per  
 Bera, Palash  
 Biffl, Stefan  
 Binemann-Zdanowicz, Aleksander  
 Bos, Rik  
 Botzer, David  
 Braganholo, Vanessa de Paula  
 Casati, Fabio  
 Catania, Barbara  
 Cavero, José María  
 Comai, Sara  
 Dahlstedt, Åsa G.  
 Damiani, Maria  
 De Backer, Manu  
 de Castro, Valeria  
 Dorneles, Carina Friedrich  
 Eckstein, Silke  
 Feijó Nadvorný, César  
 Fiedler, Gunar  
 Fons, Joan  
 Fuentes, Thayzel  
 Giurca, Adrian  
 Goethals, Frank  
 Goulão, Miguel  
 Graf, Sabine  
 Grossniklaus, Michael  
 Hadar, Irit  
 Haesen, Raf  
 Haux, Reinhold  
 Iturrioz, Jon  
 Jansen, Slinger  
 Keberle, Natalya  
 Klein, Hans-Joachim  
 Koncilia, Christian  
 Kosch, Harald  
 Kramler, Gerhard  
 Leal Musa, Daniela  
 Lehmann, Marek  
 Lemahieu, Wilfried  
 Ma, Hui  
 Martin, Maria de los A.  
 Matera, Maristella  
 Mathiak, Brigitte  
 Meinecke, Johannes  
 Michlmayr, Elke  
 Mouratidis, Haralambos  
 Muñoz, Javier  
 Neumann, Karl  
 Pantazoglou, Michael  
 Pastrana, José Luis  
 Pellens, Bram  
 Piattini, Mario  
 Pichler, Horst  
 Pilioura, Thomi  
 Plessers, Peter  
 Prévot, Laurent  
 Riaz-ud-Din, Faizal  
 Ruiz, Marta  
 Söderström, Eva  
 Sánchez, Juan  
 Sawyer, Pete  
 Scherzinger, Stefanie  
 Schmidt, Peggy  
 Schwinger, Wieland  
 Sianas, Panagiotis  
 Sierra, Iskander  
 Sneiders, Eriks  
 Somoza, Alfredo  
 Strand, Mattias  
 Sturm, Arnon  
 Torres, Victoria  
 Valderas, Pedro J.  
 Vassilakis, Costas  
 Vela, Belén  
 Wimmer, Maria A.  
 Wombacher, Andreas  
 Woo, Carson  
 Zannone, Nicola



# Table of Contents

## Keynotes

Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research .....	1
A MDA-Compliant Environment for Developing User Interfaces of Information Systems .....	16
Toward Semantic Interoperability of Heterogeneous Biological Data Sources .....	32

## Conceptual Modeling

The Association Construct in Conceptual Modelling – An Analysis Using the Bunge Ontological Model .....	33
Computing the Relevant Instances That May Violate an OCL constraint .....	48
Event-Based Modeling of Evolution for Semantic-Driven Systems .....	63

## Metamodeling

Interoperability in Meta-environments: An XMI-Based Approach .....	77
On the Notion of Consistency in Metadata Repository Systems .....	90
Using Text Editing Creation Time Meta Data for Document Management .....	105

## Databases

An Object-Relational Approach to the Representation of Multi-granular Spatio-Temporal Data ..... 119

Managing Inheritance Hierarchies in Object/Relational Mapping Tools ..... 135

BInXS: A Process for Integration of XML Schemata ..... 151

## Query Processing

Query Processing Using Ontologies ..... 167

Estimating Recall and Precision for Vague Queries in Databases ..... 187

Querying Tree-Structured Data Using Dimension Graphs ..... 201

## Process Modeling and Workflow Systems

Workflow Resource Patterns: Identification, Representation and Tool Support ..... 216

A Declarative Foundation of Process Models ..... 233

Synchronizing Copies of External Data in Workflow Management Systems ..... 248

## Requirements Engineering

Understanding the Requirements on Modelling Techniques ..... 262

A Process for Generating Fitness Measures	277
---	-----

A Concern-Oriented Requirements Engineering Model	293
---	-----

## Model Transformation

Generating Transformation Definition from Mapping Specification: Application to Web Service Platform	309
---	-----

A General Approach to the Generation of Conceptual Model Transformations	326
---	-----

Building a Software Factory for Pervasive Systems Development	342
---	-----

## Knowledge Management and Verification

Alignment and Maturity Are Siblings in Architecture Assessment	357
--	-----

Verification of EPCs: Using Reduction Rules and Petri Nets	372
--	-----

Measurement Practices for Knowledge Management: An Option Perspective	387
--	-----

## Web Services

An Ontological Approach for Eliciting and Understanding Needs in e-Services	400
--	-----

Developing Adapters for Web Services Integration	415
--	-----

Efficient: A Toolset for Building Trusted B2B Transactions  
..... 430

**Web Engineering**

Separation of Structural Concerns in Physical Hypermedia Models  
..... 446

Integrating Unnormalised Semi-structured Data Sources  
..... 460

Model Transformations in the Development of Data-Intensive Web  
Applications  
..... 475

**Software Testing**

Automated Reasoning on Feature Models  
..... 491

A Method for Information Systems Testing Automation  
..... 504

Model-Based System Testing of Software Product Families  
..... 519

**Software Quality**

Quality-Based Software Reuse  
..... 535

On the Lightweight Use of Goal-Oriented Models for Software Package  
Selection  
..... 551

Measuring IT Infrastructure Project Size: Infrastructure Effort Points  
..... 567

**Author Index** ..... 583

# Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research

Antoni Olivé

Universitat Politècnica de Catalunya  
Dept. Llenguatges i Sistemes Informàtics  
Jordi Girona 1-3, 08034 Barcelona (Catalonia)  
olive@lsi.upc.edu

**Abstract.** The goal of automating information systems building was stated in the sixties. Forty years later it is clear that the goal has not been achieved in a satisfactory degree. One of the problems has been the lack of standards in languages and platforms. In this respect, the recent efforts on standardization provide an opportunity to revive the automation goal. This is the main purpose of this paper. We have named the goal “conceptual schema-centric development” (CSCD) in order to emphasize that the conceptual schema should be the center of the development of information systems. We show that to develop an information system it is necessary to define its conceptual schema and that, therefore, the CSCD approach does not place an extra burden on developers. In CSCD, conceptual schemas would be explicit, executable in the production environment and the basis for the system evolution. To achieve the CSCD goal it is necessary to solve many research problems. We identify and comment on a few problems that should be included in a research agenda for CSCD. Finally, we show that the CSCD goal can be qualified as a grand challenge for the information systems research community.

## 1 Introduction

The goal of automating information systems building was stated in the sixties [49]. Since then, the goal has been reformulated many times, but the essential idea has remained stable: To automatically execute the specification of an information system in its production environment.

Forty years later, it is obvious that the goal has not been achieved in a satisfactory degree. In the past, there has been a lot of research in “automatic programming systems”, “automatic code generation”, “integrated computer-aided software engineering tools” and the like. Currently, several projects are pursuing the same goal, such as those described in [32, 42]. The progress has been impressive, but it is a matter of fact that, in most current information systems development and maintenance projects, the design, programming and testing activities still require substantial manual effort. In the professional information systems community, two of

the currently most popular development approaches are the Unified Process [27] and the agile methods [4]. In neither of the two, automation of the system building plays a significant role.

It is then natural to pose the question: why has the goal not been achieved?. One possible explanation could be that the goal is unachievable. However, it is difficult to single out a characteristic in the goal that could make it so hard. The goal has a precise formulation. The size of the system that should be built is not larger than that of other systems our community has built. Probably, there are several successful software systems more complex than the one that should be built for automating the building of information systems.

Another possible explanation could be that the goal has not been considered worthwhile for the research or professional communities. However, this explanation is easily invalidated by observing the amount of research done or the large effort most organizations still spend in building their information systems.

The reason why the goal has not been achieved is that a number of important problems remain to be solved [43]. Most of these problems are technical, but others are related to the lack of maturity in the information systems field, such as the lack of standards. The insufficient standardization of languages and platforms has hampered advances in the automation of systems building.

Fortunately, however, the last decade has seen the emergence of new standards related to information systems development. The progress made in standardization provides an opportunity to revive the automation goal. This is the main purpose of this paper.

We propose to call the goal “conceptual schema-centric development” (CSCD) in order to emphasize that the conceptual schema should be the center of the development of information systems. In conceptual modeling, a conceptual schema is basically the formal specification of functional requirements [29, 44]. In the next section, we briefly review the notion of conceptual schemas and analyze the nature of their relationship with information systems.

In CSCD, conceptual schemas would be explicit, executable in the production environment and the basis for the system evolution. More details are given in Section 3, where CSCD is also compared with other development approaches.

To achieve the CSCD goal it is necessary to solve many research problems. In Section 4 we identify and comment on a few problems that should be included in a research agenda for CSCD. Other relevant agendas have been presented in [12, 53].

The paper ends with an evaluation of the CSCD goal with respect to the grand challenges for computing research. According to Hoare, a grand challenge:

“represents a commitment by a significant section of the research community to work together towards a common goal, agreed to be valuable and achievable by a team effort within a predicted timescale. The challenge is formulated by the researchers themselves as a focus for the research that they wish to pursue in any case.” [25].

To be qualified as a grand challenge, a research goal has to meet a number of criteria. In section 5 we evaluate the CSCD goal according to these criteria.

## 2 Back to Basics

In this section, first we review the main functions of an information system (IS), and then we analyze the knowledge required by a particular IS to perform these functions. The analysis leads us to the definition of conceptual schemas [38].

### 2.1 Functions of an Information System

ISs can be defined from several perspectives. For the purposes of conceptual modeling, the most useful is that of the functions they perform. According to this perspective, an IS performs three main functions [6, p.74]:

- *Memory*: To maintain a consistent representation of the state of a domain.
- *Informative*: To provide information about the state of a domain.
- *Active*: To perform actions that change the state of a domain.

The memory function is passive, in the sense that it does not perform actions that directly affect users or the domain, but it is required by the other functions, and it constrains what these functions can perform.

In the informative function, the system communicates some information or commands to one or more actors. Such communication may be explicitly requested or implicitly generated when some generating condition is satisfied.

With the active function, the system performs actions that change the state of the domain. Such actions may be explicitly requested or implicitly generated when some generating condition is satisfied.

### 2.2 Knowledge Required by an Information System

To be able to perform the above functions, an IS requires some general knowledge about its domain, and knowledge about the functions it must perform. In the following, we summarize the main pieces of knowledge required by each function.

If the memory function of an IS has to maintain a representation of the state of the domain, then the IS has to know the entity and relationship types to be represented, and their current population. Some ISs may require to know that population also at some or all past time points. The entity and relationship types of interest are general knowledge about the domain, while their (time-varying) population is particular knowledge.

In conceptual modeling, we call Information Base (IB) the representation of the state of the domain in the IS. A non-temporal IB represents only the current state of the domain, while a temporal one represents also the state of the domain at any past time.

The representation of the state in the IB must be consistent. This is achieved by defining a set of conditions (called integrity constraints) and requiring that the IS satisfies them at any time. Such integrity constraints are general knowledge about the domain.

The domain state is not static. Most domains change through time, and therefore their state changes too. A domain event is a state change that consists of a set of elementary changes in the population of entity or relationship types that are considered as a single change in the domain. When the state of a domain changes, the

IB must change accordingly. The IS must know the types of the possible domain events and the effect of each event instance on the IB [39]. This is also general knowledge about the domain.

If the informative function has to communicate some information or commands on request, then the IS must know the possible request types and the output it has to communicate. On the other hand, if there are generated communications then the IS must know the generating condition and the output it has to communicate when it is satisfied. Note that this knowledge is not about the domain, but about the functions required to the IS.

In general, in order to perform the informative function the IS needs an inference capability that allows it to infer new knowledge. The inference capability requires two main elements: derivation rules and an inference mechanism. A derivation rule is general knowledge about a domain that defines a derived entity or relationship type in terms of others. The inference mechanism uses derivation rules to infer new information.

If, in the active function, the IS has to perform some action on request, then the IS must know the possible request types and the action it has to perform in each case. On the other hand, if some action must be performed when a generating condition is satisfied then the IS must know this condition and the action it has to perform. Note again that this knowledge is not about the domain, but about the functions required to the IS.

### 2.3 Conceptual Schemas

The first conclusion of the above analysis is that in order to perform its required functions, an IS must have some general knowledge about its domain and about the functions it has to perform. In the information systems field, such knowledge is called the Conceptual Schema<sup>1</sup> (CS).

Every IS embodies a CS [33, 31, 48 (p.417+)]. Without a CS, an IS could not perform any useful function. Therefore, developers need to know the CS in order to develop an IS.

Unfortunately, however, the need of CSs in IS development is often overlooked or ignored. The consequences are negative, both in theory and in practice. To help remedy the situation we propose to reformulate the need of CSs as a principle, that we propose to call the *Principle of Necessity*:

*“To develop an information system it is necessary to define its conceptual schema”.*

The principle of necessity can be seen as a consequence of the *100 Percent Principle* stated in the report [26]:

*“All relevant general static and dynamic aspects, i.e. all rules, laws, etc. of the universe of discourse should be described in the conceptual schema. The information system cannot be held responsible for not meeting those described elsewhere, including in particular those in application programs.”*

---

<sup>1</sup> In the fields of Knowledge Representation and Semantic Web the name given to an (almost) equivalent concept is Ontology [38].



The same report stated also the *Conceptualization Principle*, which says that:

*“A conceptual schema should only include conceptually relevant aspects, both static and dynamic, of the universe of discourse, thus excluding all aspects of (external or internal) data representation, physical data organization and access as well as all aspects of particular external user representation such as message formats, data structures, etc.”*

The main purpose of the activity of conceptual modeling is to elicit the CS of the corresponding IS. Given that, as we have seen, any useful IS needs a CS, we easily arrive to the conclusion that conceptual modeling is an essential activity of information systems development.

The CS must always exist, but it may take several forms, both externally and internally to the IS. Externally, the CS may be mental (exists only in the developers' heads) or explicit (written in some conceptual modeling language). When it is explicit, the CS documents the common understanding that users, analysts, and designers have about the domain and the functions imposed to the IS [29]. Internally, the CS may be diffused among the code of the IS, or be an explicit executable component. This paper advocates explicit and executable CSs.

### 3 Conceptual Schema-Centric Development

In this Section we reformulate the vision of a conceptual schema-centric development (CSCD) of information systems. We first present the characteristics of CSCD and then we compare it with some of the current development approaches.

#### 3.1 Characteristics

The conceptual schema-centric development of an information system has three main distinguishing characteristics, that we call *Explicit*, *Executable* and *Evolving Schema*.

**Explicit Schema.** Once the functions of the IS have been determined there is an explicit, complete, correct and permanently up-to-date conceptual schema, written in a domain-independent, formal and declarative language. There is a development environment with tools that facilitate the validation, testing, reuse and management of (large) schemas.

**Executable Schema.** The schema is executable in the production environment. This may be achieved by an automatic and complete transformation of the CS into software components (including the database schema) written in the languages required by the production environment, or by the use of a virtual machine running on top of that environment. In either case, the CS is the only description to be defined. All the others are internal to the system, and need not be visible externally.

According to the conceptualization principle, CSs exclude all aspects related to information presentation. Therefore, the presentation layer of an IS is outside the scope of CSCD, although it may be based on the CS [16].

**Evolving Schema.** Changes to the functions of the IS require the manual change of only its CS. The changes to this schema are automatically propagated to all system components (including the database schema and data) if needed.

### 3.2 Comparison with Other Approaches

The CSCD approach may be used either as an alternative to, or in conjunction with, other development approaches. In what follows we give examples of the two cases, taking as reference some of the currently most popular development approaches.

**Architecture-Centric.** One of the distinguishing aspects of the Unified Process [27] is that it is architecture-centric. This means that the system's architecture is used as a primary artifact for conceptualizing, constructing, managing, and evolving the system under development. An architecture-centric approach develops the functions (use cases) and the system's architecture in parallel. The rationale is that the functions drive the architecture, but at the same time the architecture guides the functions. On the other hand, an early focus on architectural issues is necessary for an iterative and incremental development process.

In the CSCD approach, the functions of the system and the CS are determined without taking into account architectural issues. The architecture of the system is either predefined or generated automatically. In either case, it is likely that the architecture is based on one or more architectural patterns widely used in the information systems field.

**Test-Driven Development.** Test-Driven Development (TDD) is one of the core practices of Extreme Programming (XP), a well-known agile method. TDD is an iterative process. Each TDD cycle is composed of five steps: (1) Write a test that defines how a part of the software should behave; (2) Run the test and see that it fails; (3) Make a little change; (4) Run the test and succeed; and (5) Refactor to remove duplication or any other problems that were introduced to get to test run [5].

CSCD can be used in conjunction with TDD. In CSCD, a test is a sequence of action requests along with comparisons of actual results with expected results. Running a test is straightforward because the schema is executable. Changes and refactoring are done at the schema level [54].

**Model-Driven Architecture.** The OMG's Model Driven Architecture (MDA) defines an approach to system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. In the MDA there are two kinds of models: Platform Independent Models (PIM) and Platform Specific Models (PSM). The PIMs provide formal specifications of the structure and behavior of the system that abstracts away technical details. A conceptual schema is a PIM. The PSMs specify the system in terms of the implementation constructs that are available in one specific implementation technology. Both PIMs and PSMs are expressed in the UML [40, 41].

One of the key features of MDA is the notion of mapping. A mapping is a set of rules and techniques used to transform a model into another one. A PIM to PSM

mapping is used when the PIM is sufficiently refined to be projected to the execution infrastructure. The projection is based on the platform characteristics. Going from the conceptual to a domain layer in the J2EE platform is an example of PIM to PSM mapping. The goal of the MDA is to make the transformation from PIMs to PSMs as automatically as possible. This goal is expected to be achieved in many cases.

When the conceptual modeling language is the UML, the CSCD approach is MDA-compliant. However, in some respects CSCD is broader than (the current version of) MDA because it includes not only the initial development of information systems, but also their evolution.

**Domain-Driven Design.** The *domain-driven design* approach [15] advocates the use of the CS as the domain layer. The idea is that if the CS is executable in the platform in which the IS is implemented, then the CS can be the domain layer. The objective is that the domain layer is literally the CS, without any mapping between both. To make such a close correspondence of CS and design possible, it is essential to use languages that serves both purposes.

The approach works well when it is acceptable to express the CS in an implementation language, such as Java. In general, this is not the case. CSs need to represent knowledge declaratively, while it must be represented procedurally in the domain layer. For example, declarative constraints, dynamic and multiple classification, operation specifications by pre/postconditions or state transition diagrams do not have a literal representation in object-oriented programming languages. Different language constructs and different languages are needed in the CS and in the domain layer.

When the conceptual modeling language used is declarative and executable in the production environment, the CSCD and the domain-driven design approaches coincide. However, in some respects CSCD is broader than domain-driven design because it includes not only the initial development of information systems, but also their evolution.

## 4 Towards a Research Agenda for CSCD

CSCD is still a research goal. There are many research problems that must be solved before CSCD is a widely used approach in the development of industrial information systems. In this Section, we identify some of the research problems in each of the three characteristics of CSCD. We emphasize here some problems related to CSCD; see [12, 53] for other relevant research agendas in conceptual modeling.

### 4.1 Explicit Schemas

**Very Large Conceptual Schemas.** The CS of a large organization may contain thousands of entity types, relationship types, constraints, etc. The development and management of (very) large CSs pose specific problems not found in small CSs. Conceptual modeling in the large is not the same as conceptual modeling in the small. The differences are similar to those observed between programming in the large and

programming in the small [13]. We need methods, techniques and tools to support designers and users in the development, reuse, evolution and understanding of large schemas.

So far, work on this topic has focused mainly on CSs for databases [1, 10, 47]. In CSCD we have to deal with ISs and we have to take into account both the structural (including constraints and derivation rules) and behavioral schemas.

**Business Rules Integration.** A business rule is a statement that defines or constrains some aspect of a business. From the information system perspective, business rules are elementary pieces of knowledge that define or constrain the contents and the change of the IB. Business rules are the main focus of a community that advocates a development approach in which the rules are explicitly defined, directly executed (for example in a rules engine) and managed [7, 45]. Given that business rules are part of CSs, we can say that, in what respects to business rules, that community already follows the CSCD approach.

It is necessary and convenient to integrate the business rules and the CSCD approaches. It should be possible to extract the rules embedded in a schema, and to present them to users and designers in a variety of ways and languages, including the natural language. Automated support for this extraction and presentation is necessary. On the other hand, it should be easy to capture a particular rule and to integrate it into the schema. Automated support for this integration is desirable.

**Complete and Correct Conceptual Schemas.** Completeness and correctness are two of the quality factors of CSs. A complete CS includes all knowledge relevant to the IS. A correct CS contains only correct and relevant knowledge. Consistency is subsumed by validity and completeness [28]. In CSCD, completeness and correctness are the prime quality factors. They can be achieved by using a very broad spectrum of approaches, including testing and verification. It should be possible to test and verify CSs at least to the same extent that has been achieved in software.

There has been some work on testing CSs [23, 32, 17, 54]. There are automatic procedures for the verification of some properties of CSs in Description Logics [9]. Model checking is being explored as an alternative verification technique [14]. In all of these topics, a lot of work remains to be done [34].

## 4.2 Executable Schemas

**Materialization of Derived Types.** In general, CSs contain many derived entity and relationship types, with their corresponding derivation rules [36]. For efficiency reasons, some of these types must be materialized. The determination of the derived types to materialize should be as automatic as possible. On the other hand, changes in the population of base types may imply changes in that of one or more materialized types. The propagation of these changes should be completely automatic.

The work done on the selection of database views to materialize in data warehouses [21] is highly relevant to the determination of the derived types to materialize in ISs. Similarly, the large body of work done in the incremental

maintenance of materialized database views [20] is highly relevant to the more general problem of change propagation in ISs.

**Enforcement of Integrity Constraints.** Most CSs contain a large number of integrity constraints [37]. The IS must enforce these constraints in an efficient way. This can be achieved in several ways [50]. The main approaches are integrity checking, maintenance and enforcement. In integrity checking and maintenance each constraint is analyzed in order to (1) determine which changes to the IB may violate the constraint; (2) generate a simplified form of the constraint, to be checked when a particular change occurs; and (3) (in maintenance) generate a repair action. In integrity enforcement each event (transaction) is analyzed in order to (1) determine which constraints could be violated by the effect of the event; and (2) generate a new version of the event effect that ensures that no constraint will be violated.

In CSCD, the analysis (whichever approach is taken) should be fully automatic and able to deal with any kind of constraint. A general method for this analysis does not exist yet. However, there has been a lot of research and development work in the enforcement of constraints in the database field, for relational, deductive and object-oriented databases [11, 51, 30]. The general method is likely to be an extension of this work.

### 4.3 Evolving Schemas

**Concepts evolution.** The most fundamental changes to a CS are adding or dropping concepts (entity, relationship or event types or states in state machines) and adding or dropping edges in the concept generalization hierarchy. These changes must be propagated to the logical schema(s) of the database(s) and to its (their) instances. The changes may affect also the existing derivation rules, constraints, event effects, etc. and, thus, the program code that implements them. Change propagation should be as automatic as possible. On the other hand, changes to the generalization hierarchy may imply a change (increase or decrease) in the population of some concepts such that some integrity constraints are violated. The IS should (efficiently) detect these violations and produce an appropriate response.

There has been an impressive amount of work on database schema evolution, focusing mainly on concepts evolution [3]. In CSCD the evolution starts at the conceptual level and it must be automatically propagated to the logical level [24]. Furthermore, more work is needed on the impact of changes to the generalization hierarchy on general integrity constraints.

**Constraints evolution.** Adding a constraint may make the IB inconsistent. Changing a constraint can be seen as a removal (which cannot lead to any inconsistency) and an addition. When a constraint is added, the IS has to check whether or not the current IB satisfies it. For very large IBs the checking may need to be efficient. If one or more fragments of the IB violate the constraint, the IS has to produce some response (reject the constraint, ignore the inconsistency, repair the fragment or handle the fragment as an exception).

In the database field, the problem of adding constraints has been studied for some particular constraints and database models [52]. In CSCD, we need to be able to deal with particular constraints (like cardinalities) but also with general constraints expressed in a conceptual modeling language, involving both base and/or derived types.

**Derivability evolution.** The derivability of entity and relationship types may change. A base type may change to derived, or the other way round. Besides, a derivation rule may change. Changing the derivability of a type may imply a change in its population, and indirectly in that of other types. If the change affects a materialized type, then it is necessary to recompute it. For large IBs the recomputation may need to be efficient. On the other hand, changing the population of a type may induce the violation of some integrity constraints. The IS should (efficiently) detect these violations and produce an appropriate response.

There has been some work on this topic [18], but much more needs to be done. A partially similar problem in the database field is that of “view adaptation” after view redefinition [22].

## 5 Is This a Grand Challenge?

CSCD is a research goal. Even if aspects of the CSCD approach may be found in current information systems development projects, there is a long way to go until its full potential may be realized in industrial projects.

In this section we assess the CSCD as a research goal. To this end, we use two sets of criteria:

- 1) The key properties of a good long-range research goal, proposed by Jim Gray [19].
- 2) The desirable properties of a research goal to qualify as a grand challenge, proposed by Tony Hoare [25].

### 5.1 A Good Long-Range Research Goal?

We evaluate first the CSCD research goal with respect to the five key properties of a good long-range research goal described in [19].

**Understandable.** The CSCD goal is very simple to state to people working in the information systems field. The goal may also be understood by the general public. The distinction between specification and implementation is sufficiently familiar, and this facilitates the understanding of the goal of easing the development and the evolution of information systems by focusing only on the specification, and leaving the implementation details to “the machine”.

**Challenging.** It is not obvious how to achieve the goal. The ideas of automatic programming, code generation, model compilers, etc. have been around for a long time, but the results obtained so far are not sufficient for the needs of most

information systems. Progress made in the automatic evolution of information systems has been even less satisfactory.

**Useful.** If the goal is achieved, the results will be useful to most organizations.

**Testable.** The goal will be achieved when we have a system that (1) facilitates the development of complete, possibly very large, declarative, tested and verified conceptual schemas; (2) automatically generates efficient information systems from these schemas, in the chosen platforms; and (3) evolves (also automatically) existing information systems from the changes to their schemas.

**Incremental.** The goal can be decomposed into intermediate milestones, in several ways. There are several ambition levels in (1) each aspect of the support to the development of CSs: completeness, coping with largeness, declarativity, testing and verification; (2) automatic generation: extent covered, platforms supported, degree of automation and level of performance; and (3) automatic evolution: types of changes supported and degree of automation.

## 5.2 A Grand Challenge?

Tony Hoare [25] suggested a set of seventeen criteria that must be satisfied by a research goal to qualify as a grand challenge. These criteria include the five key properties indicated above. The additional criteria relate to the maturity of the discipline and the feasibility of the project. In what follows we evaluate the CSCD goal with respect to the additional criteria. The order of the criteria is not significant.

**Fundamental.** Design, implementation and evolution of ISs is a fundamental concern in the information systems engineering field.

**Astonishing.** Most of the general public, and even many information systems professionals, are unaware of the possibility that computers might generate their own ISs and, above all, that changes to them can be done automatically.

**Revolutionary.** If the goal is achieved, it will lead to a radical change in the way ISs are developed and maintained in most organizations.

**Research-directed.** Significant parts of the goal can be achieved only by the methods of research. Among them there are improving conceptual modeling languages, devising methods for dealing with largeness, verification of CSs, generating efficient code from highly-declarative knowledge specification and the propagation of CS changes down to implementation.

**Inspiring.** The goal has the support from (almost) the entire information systems research community.

**International.** The goal has an international scope. Researchers from all around the world can participate in it.

**Historical.** The goal of automating information systems building was already formulated in the late sixties [35, 49] and since then it has been reformulated many times [8, 23, 42, 19]. The goal of evolving information systems from their specifications was already stated in the late seventies [46, 2] and, as before, it has been reformulated many times.

**Feasible.** The goal is now more feasible than ever. The progress recently made in standardization of languages and platforms has boosted industry interest in the goal. The basic constructs of conceptual modeling languages are already well known. It is feasible to improve current development environments. There is a lot of experience in code generation. Data persistence techniques and patterns are well known and successfully used in databases and in many data management layers. Many of the techniques of constraint enforcement, view definition and materialization, and schema evolution developed in the database field could be adapted to the broader context of ISs.

**Cooperative.** The work can be parceled out to teams working independently on (among others) conceptual modeling languages, testing and verification tools, persistence management, constraint enforcement and derived types.

**Competitive.** CSCD encourages and benefits from competition among teams. Most of the problems in the research agenda admit several solutions, with different range of application and/or efficiency.

**Effective.** The promulgation of the CSCD challenge is intended to cause a shift in the attitudes and activities of the relevant academic and professional communities. Development teams could consider the explicit definition of CSs a necessary step that needs to be done at professional quality level, using the right development environment. IS students could learn that CSs are closer to “working software” than to just “documentation”. Researchers in conceptual modeling could focus on the theoretical issues to be solved in order to build CS development environments for professional use. Researchers of constraint enforcement, materialized views and evolution in databases could accept the challenge of extending current solutions to the broader context of ISs.

**Risk-Managed.** The main critical risks to the project arise from difficulties in achieving automatic systems evolution, in particular when evolution affects existing instances. Given that ISs need to evolve very often, the CSCD goal would fail (in theory and in practice) if it were not possible to define most of the changes only at the conceptual level. An early and constant focus on the evolution issues is essential to the success of CSCD.



## 6 Conclusions

The main purpose of this paper has been to revive the goal of automating information systems building. We have named the goal “conceptual schema-centric development” (CSCD) in order to emphasize that the conceptual schema should be the center of the development of information systems.

We have shown that to develop an information system it is necessary to define its conceptual schema. In CSCD, conceptual schemas would be explicit, executable in the production environment and the basis for the system evolution. To achieve the CSCD goal it is necessary to solve many research problems. We have identified and made some comments on a few problems that should be included in a research agenda for CSCD.

Finally, we have shown that the CSCD goal can be qualified as a grand challenge for the information systems research community.

## Acknowledgements

I wish to thank the GMC group (Jordi Cabot, Jordi Conesa, Dolors Costal, Xavier de Palol, Cristina Gómez, Anna Queralt, Maria-Ribera Sancho, Ruth Raventós and Ernest Teniente) for many useful comments to previous drafts of this paper. This work has been partially supported by the Ministerio de Ciencia y Tecnología and FEDER under project TIC2002-00744.

## References

- [1] Akoka, J.; Comyn-Wattiau, I. “Entity-relationship and object-oriented model automatic clustering”. *Data & Knowledge Engineering*, 20 (1996), pp. 87-117.
- [2] Balzer, R.; Cheatham, T.E.; Green, C. “Software Technology in the 1990’s: Using a New Paradigm”, *IEEE Computer*, 1983, pp. 16-22.
- [3] Banerjee, J.; Kim, W.; Kim, H.-J.; Korth, H.F. “Semantics and Implementation of Schema Evolution in Object-Oriented Databases”. *Proc. ACM SIGMOD 1987*, pp. 311-322.
- [4] Beck, K. *Extreme Programming Explained. Embrace Change*. Addison-Wesley, 2000, 190 p.
- [5] Beck, K. *Test-Driven Development By Example*. Addison-Wesley, 2003, 220 p.
- [6] Boman, M.; Bubenko, J.A. jr.; Johannesson, P.; Wangler, B. *Conceptual Modelling*. Prentice Hall, 1997, 269 p.
- [7] BRCommunity.com (Eds.) “A Brief History of the Business Rule Approach”, *Business Rules Journal*, 6(1), January 2005.
- [8] Bubenko, J.A. jr. “Information Systems Methodology – a Research View”. In Olle, T.W.; Sol, H.G.; Verrijn-Stuart, A.A. (Eds.) *Information Systems Design Methodologies: Improving the Practice*. North-Holland, 1986, pp. 289-318.
- [9] Calvanese, D.; Lenzerini, M.; Nardi, D. “Description Logics for Conceptual Data Modeling”. In Chomicki, J.; Saake, G. (Eds.) *Logics for Databases and Information Systems*. Kluwer, 1998, pp. 229-263.
- [10] Castano, S.; de Antonellis, V.; Fugini, M.G.; Pernici, B. “Conceptual Schema Analysis: Techniques and Applications”. *ACM TODS*, 23(3), 1998, pp. 286-333.

- [11] Ceri, S.; Fraternali, P.; Paraboschi, S.; Tanca, L. "Automatic Generation of Production Rules for Integrity Maintenance". *ACM TODS*, 19(3), September 1994, pp. 367-422.
- [12] Chen, P.; Thalheim, B.; Wong, L.Y. "Future Directions of Conceptual Modeling". *Proc. ER 1997*, LNCS 1565, pp. 287-301.
- [13] DeRemer, F.; Kron, H. "Programming-in-the-Large Versus Programming-in-the-Small". *IEEE Trans. Software Eng.* 2(2) 1976, pp. 80-86.
- [14] Eshuis, R.; Jansen, D.N.; Wieringa, R. "Requirements-Level Semantics and Model Checking of Object-Oriented Statecharts". *Requirements Engineering* 7(4), 2002, pp. 243-263.
- [15] Evans, E. *Domain-Driven Design. Tackling Complexity in the Heart of Business Software*. Addison-Wesley, 2003.
- [16] Fons, J.; Pelechano, V.; Albert, M.; Pastor, O. "Development of Web Applications from Web Enhanced Conceptual Schemas". *Proc. ER 2003*, LNCS 2813, pp. 232-245.
- [17] Gogolla, M.; Bohling, J.; Richters, M. "Validation of UML and OCL Models by Automatic Snapshot Generation". *Proc UML 2003*, LNCS 2863, pp.265-279.
- [18] Gómez, C.; Olivé, A. "Evolving Derived Entity Types in Conceptual Schemas in the UML". *Proc. OOIS 2003*, LNCS 2817, pp. 33-45.
- [19] Gray, J. "What Next?. A Dozen Information-Technology Research Goals". *Journal of the ACM*, 50(1), 2003, pp. 41-57.
- [20] Gupta, A.; Mumick, I.S. *Materialized Views. Techniques, Implementations and Applications*. The MIT Press, 1999.
- [21] Gupta, H.; Mumick, I.S. "Selection of Views to Materialize in a Data Warehouse". *IEEE Trans on Knowledge and data engineering*, 17(1), January 2005, pp. 24-43.
- [22] Gupta, A.; Mumick, I.S.; Rao, J.; Ross, K.A. "Adapting Materialized Views after Redefinitions: Techniques and a Performance Study". *Information Systems* 26 (2001), pp. 323-362.
- [23] Harel, D. "Biting the Silver Bullet. Toward a Brighter Future for System Development". *Computer*, January 1992, pp. 8-20.
- [24] Hick, J-M.; Hainaut, J-L. "Strategy for Database Application Evolution: The DB-MAIN Approach". *Proc. ER 2003*, LNCS 2813, pp. 291-306.
- [25] Hoare, T. "The Verifying Compiler: A Grand Challenge for Computing Research". *Journal of the ACM*, 50(1), 2003, pp. 63-69.
- [26] ISO/TC97/SC5/WG3 *Concepts and Terminology for the Conceptual Schema and the Information Base*, J.J. Van Griethuysen (ed.), March 1982.
- [27] Jacobson, I.; Booch, G.; Rumbaugh, J. *The Unified Software Development Process*. Addison-Wesley, 1999, 463 p.
- [28] Lindland, O.I.; Sindre, G.; Solvberg, A. "Understanding Quality in Conceptual Modeling". *IEEE Software*, March 1994, pp. 42-49.
- [29] Loucopoulos, P. "Conceptual Modeling". In Loucopoulos, P.; Zicari, R. (Eds). *Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development*. Wiley, 1992, pp. 1-26.
- [30] Mayol, E.; Teniente, E. "Consistency preserving updates in deductive databases", *Data & Knowledge Eng.*, 47(1), 2003, pp. 61-103.
- [31] Mays, R.G. "Forging a silver bullet from the essence of software". *IBM Systems Journal*, 33(1), 1994, pp. 20-45.
- [32] Mellor, S.J.; Balcer, M.J. *Executable UML. A Foundation for Model-Driven Architecture*. Addison-Wesley, 2002, 368 p.
- [33] Mylopoulos, J. "The Role of Knowledge Representation in the Development of Specifications". *Proc IFIP-86*, North-Holland, 1986, pp. 317-319.

- [34] Mylopoulos, J. "Information Modeling in the Time of the Revolution". *Information Systems* 23(3/4), 1998, pp. 127-155.
- [35] Nunamaker, J. F. "A methodology for the design and optimization of information processing systems". *Proc. Spring Joint Computer Conference*, 1971, pp. 283-294.
- [36] Olivé, A. "Derivation Rules in Object-Oriented Conceptual Modeling Languages". *Proc. CAiSE 2003, LNCS 2681*, pp. 404-420.
- [37] Olivé, A. "Integrity Constraints Definition in Object-Oriented Conceptual Modeling Languages". *Proc. ER 2003, LNCS 2813*, pp. 349-362.
- [38] Olivé, A. "On the Role of Conceptual Schemas in Information Systems Development". *Proc. Ada Europe 2004, LNCS 3063*, pp. 16-34.
- [39] Olivé, A. "Definition of Events and Their Effects in Object-Oriented Conceptual Modeling languages". *Proc. ER 2004, LNCS 3288*, pp. 136-149.
- [40] OMG. "Model Driven Architecture (MDA)". Document number ormsc/2001-07-01. 2001.
- [41] OMG. "MDA Guide Version 1.0.1". *OMG Document omg/2003-06-01*. 2003.
- [42] Pastor, O.; Gómez, J.; Insfrán, E.; Pelechano, V. "The OO-method approach for information systems modeling: from object-oriented conceptual modeling to automated programming", *Information Systems*, 26(7), 2001, pp. 507-534.
- [43] Rich, C.; Waters, R.C. "Automatic Programming: Myths and Prospects". *Computer*, August 1988, pp. 40-51.
- [44] Rolland, C.; Prakash, N. "From conceptual modeling to requirements engineering". *Annals of Software Engineering*, 10(2000), pp. 151-176.
- [45] Ross, R.G. (Ed.) "The Business Rules Manifesto". *Business Rules Group*. Version 2.0, November 2003.
- [46] Ruth, G. R. "Automatic programming: Automating the software system development process", *Proceedings of the 1977 annual conference*, pp: 174 – 180.
- [47] Shoval, P.; Danoch, R.; Balabam, M. "Hierarchical entity-relationship diagrams: the model, method of creation and experimental evaluation", *Requirements Eng.* (2004) 9: 217-228.
- [48] Sowa, J.F. *Knowledge Representation. Logical, Philosophical and Computational Foundations*. Brooks/Cole, 2000, 594p.
- [49] Teichroew, D.; Sayani, H. "Automation of System Building", *Datamation*, 17(16), August 1971, pp. 25-30.
- [50] Teniente, E.; Urpí, T. "On the abductive or deductive nature of database schema validation and update processing problems". *Theory and Practice of Logic Programming* 3(3), pp. 287-327 (2003).
- [51] Thalheim, B. *Entity-Relationship Modeling. Foundations of Database Technology*. Springer, 2000, 627 p.
- [52] Türker, C.; Gertz, M. "Semantic integrity support in SQL:1999 and commercial (object-) relational database management systems". *The VLDB Journal*, 10, 2001, pp. 241-269.
- [53] Wand, Y.; Weber, R. "Research Commentary: Information Systems and Conceptual Modeling – A Research Agenda". *Information Systems Research*, 13(4), December 2002, pp. 363-376.
- [54] Zhang, Y. "Test-Driven Modeling for Model-Driven Development", *IEEE Software*, September/October 2004, pp. 80-86.

# A MDA-Compliant Environment for Developing User Interfaces of Information Systems

Jean Vanderdonckt

Université catholique de Louvain (UCL), School of Management (IAG),  
Information Systems Unit (ISYS), Belgian Lab. of Computer-Human Interaction (BCHI),  
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium  
vanderdonckt@isys.ucl.ac.be  
<http://www.isys.ucl.ac.be/bchi/>

**Abstract.** To cope with the ever increasing diversity of markup languages, programming languages, tool kits and interface development environments, conceptual modeling of user interfaces could bring a framework for specifying, designing, and developing user interfaces at a level of abstraction that is higher than the level where code is merely manipulated. For this purpose, a complete environment is presented based on conceptual modeling of user interfaces of information systems structured around three axes: the models that characterize a user interface from the end user's viewpoint and the specification language that allows designers to specify such interfaces, the method for developing interfaces in forward, reverse, and lateral engineering based on these models, and a suite of tools that support designers in applying the method based on the models. This environment is compatible with the Model-Driven Architecture recommendations in the sense that all models adhere to the principle of separation of concerns and are based on model transformation between the MDA levels. The models and the transformations of these models are all expressed in UsiXML (User Interface eXtensible Markup Language) and maintained in a model repository that can be accessed by the suite of tools. Thanks to this environment, it is possible to quickly develop and deploy a wide array of user interfaces for different computing platforms, for different interaction modalities, for different markup and programming languages, and for various contexts of use.

## 1 Introduction

Today, the development of the User Interface (UI) of interactive applications, whether they are an information system or a complex, possibly safety-critical system, poses a series of unprecedented challenges due to the multiplication of several variables [10]:

- *Diversity of users:* the end users of a same interactive application could no longer be considered similar as they exhibit various skills, capabilities, levels of experience and preferences that should be reflected in the UI. Instead of having one single UI for all users, a family of different UIs should be developed to cope with the differences of multiple categories of users, including those who are impaired.
- *Richness of cultures:* when an interactive application is going to be global, its UI cannot remain the same for all languages, countries, and cultures. Rather, it can be

submitted to a process of localization to tailor the UI to particular constraints or to a process of globalization to adapt the UI to the largest population possible.

- *Complexity of interaction devices and styles*: Human-Computer Interaction (HCI) is an area known for dealing with a wide variety of interaction devices (e.g., bi-manual mouse, 3D pointers, laser pointer, phantom) and styles (3D tracking, eye tracking, gesture recognition, speech recognition and synthesis). The handling of events generated by these devices and their sound incorporation into an interaction style requires many programming skills that often go beyond the classical capabilities of an average developer of an information system. Even more, when several modalities are combined, as in a multimodal application, this complexity is decupled. Same for virtual reality, augmented reality and mixed reality applications.
- *Heterogeneous computing platforms*: the market of computing platforms is submitted to a constant introduction of new computing platforms, each one coming with a new set of constraints to be imposed on the UI that should run on it. For instance, a significant constraint is the screen resolution and the interaction capabilities that largely vary depending on the computing platform: mobile phone, smartphone, Pocket PC, Blackberry, Handbag PC, Tablet PC, interactive kiosk, laptop, desktop, multi-displays workstation, projected UI, wall screen. All these computing platforms do not necessarily run the same operating system and the UI is not necessarily developed with the same markup language (e.g., WML, cHTML, HTML, DHTML, VoiceXML, X+V, VRML97, X3D) or programming language (e.g., Visual Basic, C++, Java, C#). Even when a same language could be used, several peculiarities are present on each platform, thus preventing the developer from reusing code from one platform to another. The typical end user is using today at least three platforms, sometimes with some synchronization between.
- *Multiplicity of working environments*: end users are nowadays confronted to a series of different physical environments where they are supposed to work with the same reliability and efficiency. But when the environment becomes more constraining, e.g., in stress, in noise, in light, in availability of network resources, the UI is not necessarily adapted to these variations.
- *Multiplicity of contexts of use*: if a given context of use is defined as a particular user category working with a given platform in a specific environment, then the array of potential contexts of use explodes. Of course, not all the contextual variations should be considered and interpreted in a significant change of the UI, but at least a reasonable amount of differences exist for context-aware applications. From a user's perspective, various scenarios may occur [1,3]:
  1. Users may move between different computing platforms whilst involved in a task: when buying a movie on DVD a user might initially search for it from her desktop computer, read the reviews of the DVD on a PDA on the train on the way home from work, and then order it using a WAP-enabled mobile phone.
  2. The context of use may change whilst the user is interacting: the train may go into a dark tunnel so the screen of the PDA dims, the noise level will rise so volume of audio feedback increases so it can still be heard.

3. Users may want to collaborate on a task using heterogeneous computing platforms: the user decides to phone up a friend who has seen the movie and look at the reviews with her, one person using WebTV and the other using a laptop, so the same information is presented radically differently.
- *Multiplicity of software architectures*: due to the above variations, the UI should be developed with dedicated software architecture in mind (e.g., [5]) that is explicitly addressing the variations considered as for mobile computing, ubiquitous computing, context-aware applications (e.g., [1]).

Therefore, it is rather difficult to obtain a UI that addresses these variations while avoiding reproducing multiple UIs for different contexts of use without factoring out the common parts. In addition, when in the past, it was possible to code a UI by hand, today this empirical, opportunistic approach is no longer viable. All these reasons and others stem for a methodology for *User Interface Engineering*. This discipline is located midway between Software Engineering (SE), Human-Computer Interaction (HCI) and Human Factors (HF). Its primary goal is to develop a methodology for developing the UI throughout the development life cycle that can be articulated with traditional SE concepts. This methodology consists of:

1. A series of models pertaining to various facets of the UI such as: task, domain, user, presentation, dialog, platform, context of use, etc. These models will be defined in Section 2 and located on a reference framework. These models are uniformly and univocally expressed according to a single Specification Language, described in Section 3.
2. A step-wise method towards Computer-Aided Design of User Interfaces (CADUI) based on any combination of the above models. Section 4 will define this method by combination of models and model transformations so as to be compliant with the Model-Driven Architecture, to support Model-Driven Development (MDD).
3. A suite of software engineering tools that supports the designer and the developer during the development life cycle according to the method. A subset of these tools will be illustrated in Section 5.

Section 6 will summarize the main benefits of the MDA-compliant environment.

## 2 Models

Our methodology is explicitly based on the Cameleon Reference Framework [3], which defines UI development steps for multi-context interactive applications. Its simplified version, reproduced in Fig. 1, structures development processes for two contexts of use into four development steps (each development step being able to manipulate any specific artifact of interest as a model or a UI representation):

1. *Final UI* (FUI): is the operational UI i.e. any UI running on a particular computing platform either by interpretation (e.g., through a Web browser) or by execution (e.g., after compilation of code in an interactive development environment).

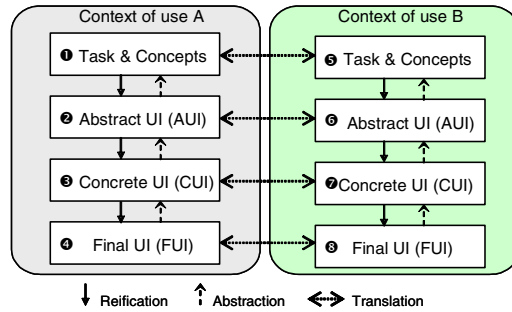


Fig. 1. The Simplified User Interface Reference Framework [3]

2. *Concrete UI (CUI)*: concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) [19] so as to define widgets layout and interface navigation. It abstracts a FUI into a UI definition that is independent of any computing platform. Although a CUI makes explicit the final Look & Feel of a FUI, it is still a mock-up that runs only within a particular environment (Fig. 2). A CUI can also be considered as a reification of an AUI at the upper level and an abstraction of the FUI with respect to the platform. For example, in Envir3D [18], the CUI consists of a description of traditional 2D widgets with mappings to 3D by relying on different mechanisms when such a mapping is possible.
3. *Abstract UI (AUI)*: defines abstract containers and individual components [12,13], two forms of Abstract Interaction Objects [19] by grouping subtasks according to various criteria (e.g., task model structural patterns, cognitive load analysis, semantic relationships identification), a navigation scheme between the container and selects abstract individual component for each concept so that they are independent of any modality. An AUI abstracts a CUI into a UI definition that is independent of any modality of interaction (e.g., graphical interaction, vocal interaction, speech synthesis and recognition, video-based interaction, virtual, augmented or mixed reality). An AUI can also be considered as a canonical expression of the rendering of the domain concepts and tasks in a way that is independent from any modality of interaction. An AUI is considered as an abstraction of a CUI with respect to interaction modality. At this level, the UI mainly consists of input/output definitions, along with actions that need to be performed on this information (Fig. 3). The AUI is mainly based on the Canonical Abstract Prototypes [4].
4. *Task & Concepts (T&C)*: describe the various user's tasks to be carried out and the domain-oriented concepts as they are required by these tasks to be performed. These objects are considered as instances of classes representing the concepts. Fig. 4 represents a potential task model representing the end user's viewpoint for an Internet Radio Player, based on LOTOS operators.

This framework also exhibits three types of transformation types: (1,2) *Abstraction* (respectively, *Reification*) is a process of eliciting artifacts that are more abstract (respectively, concrete) than the artifacts that serve as input to this process. Abstraction

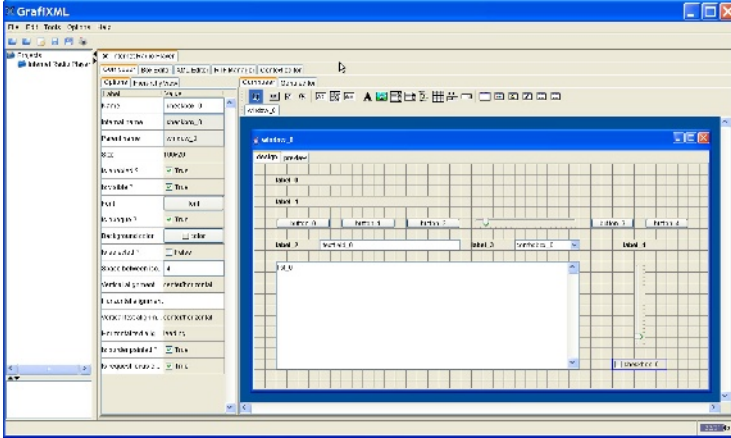


Fig. 2. A concrete UI of an Internet Radio Player [15] in GrafXML

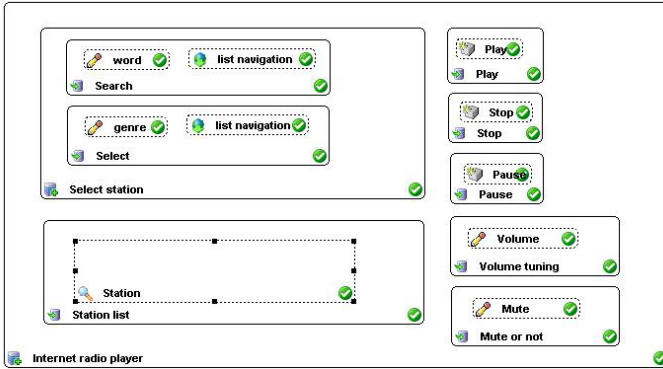
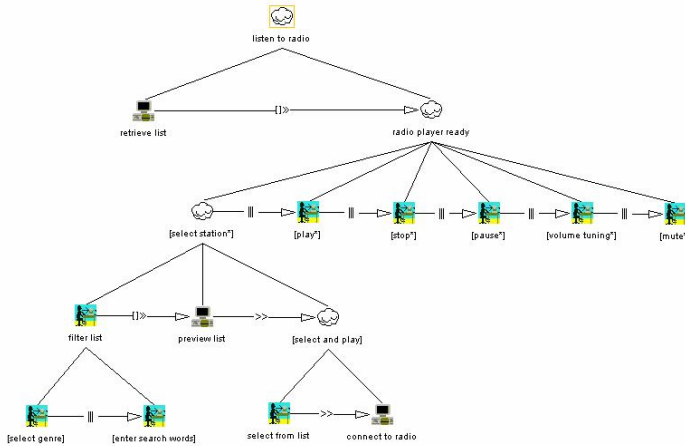


Fig. 3. A Abstract User Interface of an Internet Radio Player [15] in IdealXML [16]

is the opposite of reification. (3) *Translation* is a process that elicits artifacts intended for a particular context of use from artifacts of a similar development step but aimed at a different context of use. Therefore, when there is a need to switch from one context of use (e.g., one platform) to another one (e.g., another platform), the development process can cope with adaptation to the new context of use at any level of abstraction. Of course, the higher the level of abstraction the adaptation is performed, the more flexibility we have in the resulting processes.

With respect to this framework, multi-path UI development [12,13] refers to a UI engineering method and tool that enables a designer to (1) start a development activity from any entry point of the reference framework (Fig. 1), (2) get substantial support in the performance of transformation types and their combinations as found in Fig. 1. Several interesting development paths can be expressed on this framework since not





**Fig. 4.** A task model of an Internet Radio Player [15] in IdealXML [16]

all steps should be achieved in a sequential ordering dictated by the levels. Instead, locating *what* steps are performed, when, from which entry point and toward what subsequent step are important. According to Fig. 1, *transcoding* tools start with a FUI for a source context of use (4) and transforms it into another FUI for a target context (5). Similarly, *portability* tools start with a CUI for a source context (5) and transforms it into another CUI for another context (7), that in turn leads to a new FUI for that context (6). To overcome shortcomings identified for these tools, there is a need to raise the level of abstraction by working at the AUI level. *UI Reverse Engineering* abstracts any initial FUI (4) into concepts and relationships denoting a AUI (2), which can then be translated into a new AUI (6) by taking into account constraints and opportunities for the new context. *UI Forward Engineering* then exploits this AUI (6) to regenerate a new AUI adapted to this platform, by recomposing the CUI (7) which in turn is reified in an executable FUI (8). In general, UI reverse engineering is any combination of abstraction relationships starting from a FUI (4), a CUI (5) or an AUI (2). UI forward engineering is any combination of reification relationships starting from T&C, AUI or CUI. Similarly, *UI Lateral Engineering* is responsible for applying any translation at any level of abstraction to transform the artifacts existing at the level where we are for another context of use at the same level. For instance, when a designer has already designed a UI for, let us say a desktop, and wants to design a corresponding UI for a Pocket PC, she may need to apply *Graceful Degradation* techniques [7], which consist of a series of transformations to support the translation from the desktop to the PocketPC, while taking into account the constraints imposed by the new platform.

So far, to support conceptual modeling of UIs and to describe UIs at various levels of abstractions, the following models have been involved (Fig. 5) [13]:

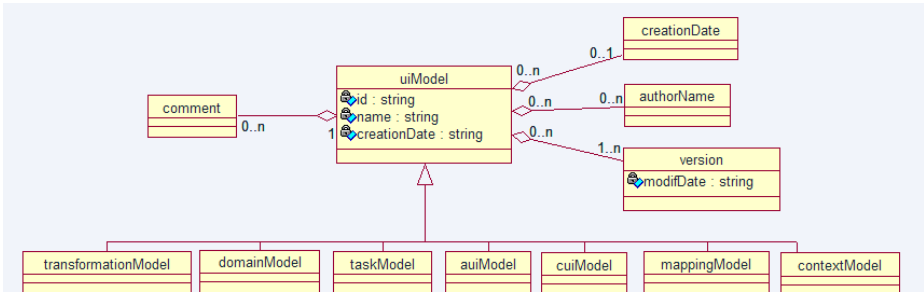


Fig. 5. The collection of models for specifying a user interface [12,13]

- **taskModel**: is a model describing the interactive task as viewed by the end user interacting with the system. A task model represents a decomposition of tasks into sub-tasks linked with task relationships. Therefore, the decomposition relationship is the privileged relationship to express this hierarchy, while temporal relationships express the temporal constraints between sub-tasks of a same parent task.
- **domainModel**: is a description of the classes of objects manipulated by a user while interacting with a system. Typically, it could be a UML class diagram, an entity-relationship-attribute model, or an object-oriented model.
- **mappingModel**: is a model containing a series of related mappings between models or elements of models. A mapping model serves to gather a set of inter-model relationships that are semantically related. It expresses reification, abstraction, and translation. In addition, other mappings [13] are defined and could be defined.
- **contextModel**: is a model describing the three aspects of a context of use in which a end user is carrying out an interactive task with a specific computing platform in a given surrounding environment [3]. Consequently, a context model consists of a *user model*, a *platform model* [5], and an *environment model*.
- **auiModel**: is the model describing the UI at the abstract level as previously defined.
- **cuiModel**: is the model describing the UI at the concrete level as previously defined.
- **uiModel**: is the topmost superclass containing common features shared by all component models of a UI. A uiModel may consist of a list of component model sin any order and any number, such as task model, a domain model, an abstract UI model, a concrete UI model, mapping model, and context model. A user interface model needs not include one of each model component. Moreover, there may be more than one of a particular kind of model component.

The conceptual modeling activities that reached to the meta-model of the cuiModel represented a significant amount of work and is therefore detailed further in the next subsection. The transformation model is the only remaining model: as such, it is defined in the second subsection.

## 2.1 The cuiModel

A CUI is assumed to be described without any reference to any particular computing platform or toolkit of that platform. For this purpose, a CUI model consists of a hierarchical decomposition of CIOs. A *Concrete Interaction Object* (CIO) is defined as any UI entity that users can perceive such as text, image, animation and/or manipulate such as a push button, a list box, or a check box [12,13,19]. A CIO is characterized by various attributes such as, but not limited to: *id*, *name*, *icon*, *content*, *defaultContent*, *defaultValue*.

Since a CIO is independent of any computing platform, we do not know yet which interaction modality is used on that platform. Therefore, each CIO can be sub-typed into sub-CIOs depending on the interaction modality chosen: *graphicalCIO* for GUIs, *auditoryCIO* for vocal interfaces, *3DCIO* for 3D UIs, etc. In this paper, we focus on graphical CIO since they form the basic elements of a traditional 2D GUI or a 3D, virtual UI. Each *graphicalCIO* inherits from the above CIO properties and has specific attributes such as: *isVisible*, *isEnabled*, *fgColor* and *bgColor* to depict foreground and background colors, etc.

Each *graphicalCIO* is then sub-typed into one of the two possible categories (Fig. 6): *graphicalContainer* for all widgets containing other widgets such as page, window, frame, dialog box, table, box and their related decomposition or *graphicalIndividualComponent* for all other traditional widgets that are typically found in such containers. A *graphicalIndividualComponent* cannot be further decomposed. The model supports a series of widgets defined as *graphicalIndividualComponents* such as: *textComponent*, *videoComponent*, *imageComponent*, *imageZone*, *radioButton*, *toggleButton*, *icon*, *checkbox*, *item*, *comboBox*, *button*, *tree*, *menu*, *menuItem*, *drawingCanvas*, *colorPicker*, *hourPicker*, *datePicker*, *filePicker*, *progressionBar*, *slider*, and *cursor*.

Thanks to this progressive inheritance mechanism, every final elements of the CUI inherits from the upper properties depending on the category they belong to. The properties that have been chosen to populate the CUI level have been decided because they belong to the intersection of property sets of major toolkits and window managers, such as Windows GDI, Java AWT and Swing, HTML. Of course, only properties of high common interest were kept. In this way, a CIO can be specified independently from the fact that it will be further rendered in HTML, VRML or Java. This quality is often referred to as the property of *platform independence*.

Similar abstractions exist for auditory, vocal, 3D, virtual, and augmented reality interfaces.

## 2.2 The TransformationModel

Graph Transformation (GT) techniques were chosen to formalize explicit transformations between any pair of models [13] (except from the FUI level), because it is (1) **Visual**: every element within a GT based language has a graphical syntax; (2) **Formal**: GT is based on a sound mathematical formalism (algebraic definition of graphs

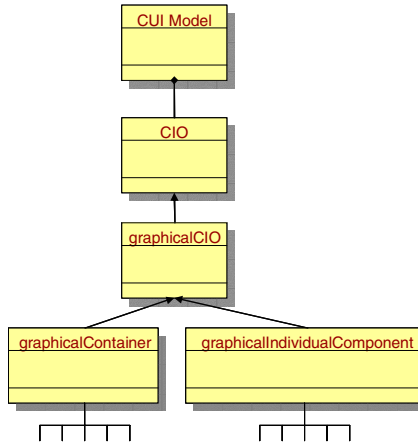


Fig. 6. Decomposition of a CUI model into concepts

and category theory) and enables verifying formal properties on represented artefacts; (3) **Seamless**: it allows representing manipulated artefacts and rules within a single formalism. Furthermore, the formalism applies equally to all levels of abstraction of uiModel (Fig. 1). The model collection (Fig. 5) is structured according to the four basic levels of abstractions defined in the Cameleon Reference Framework [3] that is intended to express the UI development life cycle for context-sensitive interactive applications. The FUI level is the only model that cannot be supported by graph transformations because it would have supposed that any markup or programming language to be supported should have been expressed in a meta-model to support transformations between meta-models: the one of the initial language to the one of our specification language (Section 3). It was observed that to address this problem, the powerfulness of GT techniques was not needed and surpassed by far other experienced techniques, such a derivation rules [2].

### 3 Specification Language

In order to specify the different UI aspects and related models, a specification language is needed that allow designers and developers to exchange, communicate, and share fragments of specifications and that enables tools to operate on these specifications. Therefore, a User Interface Description Language (UIDL) is required that satisfies at least the following important requirements: *software processability* (the UIDL should be precise enough to enable computational processing of the specifications by an automaton, in particular to interpret or execute them), *expressiveness* (the UIDL should be expressive enough to support common SE techniques such as model derivation, model transformation, mapping), *standard format* (the UIDL should be expressed in a format that maximizes its exchange among stakeholders), *human readability* (the UIDL should be as much as possible legible and understandable by a hu-

man agent), *concision* (the UIDL should be compact enough to be easily exchanged among interested parties). Since software processability and expressiveness are the two most important requirements, trade-offs could be accepted to satisfy those two requirements, while decreasing the value of the two other requirements.

To address the above requirements, we introduced UsiXML (which stands for User Interface eXtensible Markup Language – <http://www.usixml.org>), a XML-compliant markup language that describes the UI for multiple contexts of use such as Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory and Vocal User Interfaces, Virtual Reality, and Multimodal User Interfaces:

- UsiXML is primarily intended for non-developers, such as analysts, specifiers, designers, human factors experts, project leaders, and novice programmers.
- UsiXML can be equally used by experienced designers and developers.
- Thanks to UsiXML, non-developers can shape the UI of any new interactive application by specifying, describing it in the UIDL, without requiring programming skills usually found in markup languages and programming languages.
- UsiXML consists of a declarative UIDL that capturing the essence of what a UI is or should be independently of physical characteristics.
- UsiXML describes at a high level of abstraction the constituting UI elements of an application: widgets, controls, containers, modalities, and interaction techniques.
- UsiXML allows cross-toolkit development of an interactive application.
- A UI of any UsiXML-compliant application runs in all toolkits that implement it: compilers and interpreters.
- UsiXML supports *device independence*: a UI can be described in a way that remains autonomous with respect to the devices used in the interactions (e.g., mouse, screen, keyboard, voice recognition system). In case of need, a reference to a particular device can be incorporated.
- UsiXML supports *platform independence*: a UI can be described in a way that remains autonomous with respect to the various existing computing platforms (e.g., mobile phone, Pocket PC, Tablet PC, kiosk, laptop, desktop, and wall screen). In case of need, a reference to a particular computing platform can be incorporated.
- UsiXML supports *modality independence*: a UI can be described in a way that remains independent of any interaction modality (e.g., graphical interaction, vocal interaction, 3D interaction, virtual reality interaction). In case of need, a reference to a particular modality can be incorporated.
- UsiXML allows reusing elements previously described in anterior UIs to compose a UI in new applications.

On the other hand, it is not supposed to cover all features of all types of UI:

- UsiXML does not want to introduce yet another language for UI implementation. Instead, it proposes the integration of some of these formats: cHTML, WML, HTML, XHTML, VoiceXML, VRML, Java, C++,.... It is up to the underlying implementation to support the transformation of UsiXML into such a format.
- UsiXML does not describe the low-level details of elements involved in the various modalities, such as operating system attributes, events, and primitives.

- UsiXML cannot be rendered nor executed by its own: it relies on an implementation in any third-party rendering engine.
- UsiXML does not want to support all attributes, events, and primitives of all widgets existing in nearly all toolkits. Instead, it is intended to support a common subset of them that is believed to be representative and significant.

For the moment, the semantics of UsiXML are defined according to the above models and relationships in terms of a UML class diagram, representing the meta-model of the UIDL. All class diagrams are maintained in Rational Rose and lead to the definition of ML schemas that are available at <http://www.usixml.org/index.php?view=page&idpage=5> thanks to a series of systematic transformations. Therefore, any UI specification is expressed in UsiXML that is in turn compliant with the XML schemas.

## 4 Method

So far, many attempts to establish a comprehensive model-based approach for developing the UI have been launched [10,11,20,21], but only a few of them is MDA-compliant: form information related task (what are the actions carried out by the user), domain (what are the objects manipulated in this task), user (who is the user), platform (what is the computing platform), environment (in which environment is the user working), the presentation, the dialog, the help, the tutorial of one or many UIs should be derived. Today, no consensus has been reached and no method has really emerged from these initiatives, namely by lack of standardization. Since 1997, the Object Management Group (OMG – [www.omg.org](http://www.omg.org)) has launched an initiative called Model Driven Architecture (MDA) to support the development of complex, large, interactive software systems providing a standardized architecture with which:

- Systems can easily evolve to address constantly evolving user requirements.
- Old, current and new technologies can be harmonized.
- Business logic can be maintained constant or evolving independently of the technological changes.
- Legacy systems can be integrated and unified with new systems.

In this approach, models are applied in all steps of development up to a target platform, providing source code, deployment and configuration files,... MDA has been applied to many kinds of business problems and integrated with a wide array of other common computing technologies, including the area of UIs.

In MDA, a systematic method is recommended to drive the development life cycle to guarantee some form of quality of the resulting software system. Four principles underlie the OMG's view of MDA [14]:

1. Models are expressed in a well-formed unified notation and form the cornerstone to understanding software systems for enterprise scale information systems. The semantics of the models are based on meta-models.
2. The building of software systems can be organized around a set of models by applying a series of transformations between models, organized into an architectural framework of layers and transformations: model-to-model transformations

support any change between models while model-to-code transformation are typically associated with code production, automated or not.

3. A formal underpinning for describing models in a set of meta-models facilitates meaningful integration and transformation among models, and is the basis for automation through software.
4. Acceptance and adoption of this model-driven approach requires industry standards to provide openness to consumers, and foster competition among vendors.

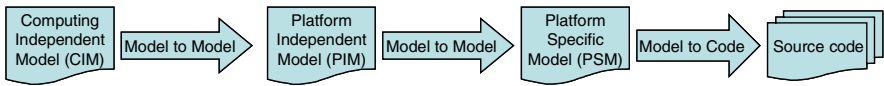
Our UI engineering methodology, based on UsiXML (Section 3), is compliant with these four principles in the following way:

1. All involved models are expressed in UsiXML, a well-formed UIDL based on XML schema. The semantics of the UsiXML models are based on meta-models expressed in terms of UML class diagrams, from which the XML schema definition are derived. Right now, there is no automation between the initial definition of the semantics and their derivation into XML schemas. Only a systematic method is used for each new release.
2. All model-to-model transformations are themselves specified in UsiXML to keep only one UIDL throughout the development life cycle. Model-to-code transformation is ensured by appropriate tools (see Section 5) that produce code for the target context of use or platform. For reverse engineering, code-to-model transformations are mainly achieved by *derivation rules* that are based on the mapping between the meta-model of the source language (e.g., HTML, WML, VoiceXML) and the meta-model of the target language (i.e., here UsiXML). So far, derivation rules have been proved powerful enough to express reverse engineering rules for UI, while keeping a relative concision.
3. All transformations are explicitly defined based on a series of predefined semantic relationship and a set of three primitive ones (abstraction, reification, and translation). The transformation model could itself contain transformation rules.
4. The last principle, i.e., the standardization process, is only on the way. Only the future will tell us whether a wide adoption of the above techniques will be effective.

In addition to the adherence of the basic MDA principles, our UI engineering methodology classifies the involved models in a similar way. Fig. 7 graphically depicts the distribution of models according to the OMG paradigm of MDA and their UsiXML counterpart: task and domain models (T&C) are considered as the Computing Independent Model (CIM) as they are stated independently of any implementation of any interactive systems. Such models could be specified for virtually any UI type. The Platform Independent Model (PIM) is interpreted as the Abstract UI model (AUI) in UsiXML in the sense that it is independent of any interaction modality: at this level, we do not know yet whether the UI will be graphical, modal, virtual or multimodal. The Platform Specific Model (PSM) is interpreted as the Concrete UI model (CUI) in UsiXML in the sense that it is independent of any vocabulary of markup and programming languages. At this level, we have already chosen what kind on interaction modality will be exploited, but we do not know yet which physical computing platform will run the UI. This is why it should be believed that the CUI is not platform

specific. Only some aspects of the target platform are selected, the platform being modeled itself in the platform model. In contrast with other MDA-compliant architectures, the present one can either render the UI directly (by interpretation) or automatically generate code (by generation) that will be compiled, linked with the rest of the application, and executed. Therefore, there is no ‘model-to-code’ transformation per se. Rather, different tools produce different results from the UsiXML specifications. For other ‘model-to-model’ transformations, graph transformation techniques are exploited throughout the development life cycle to maintain consistency.

MDA Components



Techniques proposed based on UsiXML

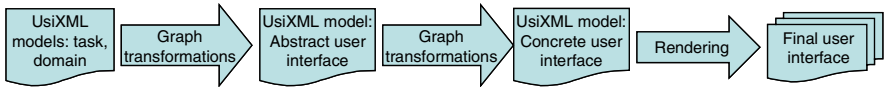


Fig. 7. The distribution of UsiXML models according to the MDA classification

## 5 Tools

Fig. 7 mainly depicts the forward engineering of UIs. As our UI engineering methodology is not restricted to this development path, other paths could be followed as well [13]. Fig. 8 somewhat generalizes the development life cycle, but for one context of use at a time. To support the application of a particular development path, a suite of tools has been developed and is currently being expanded (see <http://www.usixml.org/index.php?view=page&idpage=20> for more information). The most significant tools belonging to this suite are (Fig. 8):

- TransformiXML is a Java-based application that is responsible for defining, storing, manipulating, and executing productions contained in graph grammars to support graph transformations (model-to-model transformations)
- IdealXML [16] is a Java-based application containing the graphical editor for the task model, the domain model, and the abstract model. It can also establish any mapping between these models either manually (by direct manipulation) or semi-automatically (by calling TransformiXML).
- KnowiXML [8] consists of an expert system based on Protégé that automatically produces several AUIs from a task and a domain models for various contexts.
- GrafiXML [12] is the most elaborate UsiXML high-fidelity editor with editing of the CUI, the context model and the relationships between. It is able to automatically generate UI code in HTML, XHTML, XUL and Java thanks to a series of plug-ins.



- VisiXML is a Microsoft Visio plug-in for drawing in mid-fidelity graphical UIs, that is UIs consisting exclusively of graphical CIOs. It then exports the UI definition in UsiXML at the CUI level to be edited by GrafiXML or another editor.
- SketchiXML [6] consists of a Java low-fidelity tool for sketching a UI for multiple users, multiple platforms (e.g., a Web browser, a PDA), and multiple contexts of use. It is implemented on top of Jack agent system.
- FormiXML is a Java editor dedicated to interactive forms with a smart system of copy/paste techniques to support reusability of components. It automatically generates the complete UI code for Java/Swing.
- Several renderers are currently being implemented: FlashiXML opens a CUI UsiXML file and renders it in Flash, QtkXML in the Tcl/Tk environment, and JaviXML for Java.
- VisualiXML [17] personalizes a UI and produces one thanks to generative programming techniques for Visual C++ V6.0.
- ReversiXML [2] opens a HTML file and reverse engineers it into UsiXML at both the CUI and AUI levels.

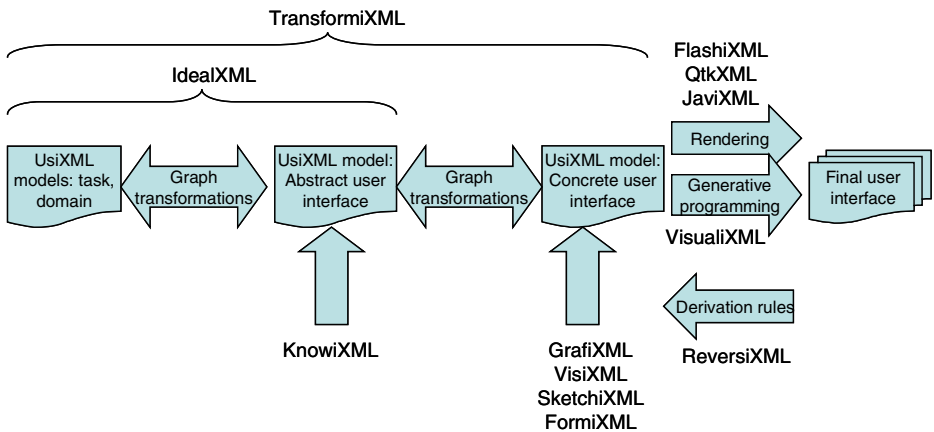


Fig. 8. The suite of UsiXML tools structured according to the MDA classification.

## 6 Conclusion

In this paper, we have introduced a UI Engineering methodology articulated on three axes: models and their specification language, method, and tools that support the method based on the underlying models. All aspects are stored in UsiXML ([www.usixml.org](http://www.usixml.org)) files that can be exchanged, shared, and communicated between stakeholders (designers, developers, and end users). It has been demonstrated that the global methodology adheres to the principles of MDA and is therefore compliant, except for the standardization process which is ongoing. It is worth to note that this environment has been largely studied and scrutinized for user interfaces of information systems that are equipped with different types of interfaces (graphical, vocal,

virtual, and multimodal mainly) on different types of computing platforms. We believe that a significant portion of UsiXML models could be equally used for the UI of more sophisticated interactive systems (e.g., [9]), like safety-critical systems, industrial supervision systems, etc. But this is matter of more extensive study. It is likely that the model transformations will be more complex to discover and to apply.

## Acknowledgements

The author would like to thank all members of the Belgian Lab. of Computer-Human Interaction (BCHI) for their fruitful involvement in the UsiXML initiative. We gratefully acknowledge the support of the following projects: the SIMILAR network of excellence (the European research task force creating human-machine interfaces similar to human-human communication – <http://www.similar.cc>), the SALAMANDRE research project (User Interfaces for Mobile and Multi-platform Interactive Systems, Initiatives III Research Program, DGTRE, Ministry of Walloon Region), the REQUEST research project (WIST-Wallonie Information Société Technologies program under convention n°031/5592), the Cameleon project (Context Aware Modelling for Enabling and Leveraging Effective interaction – FP6-IST5-2000, <http://giove.cnuce.cnr.it/cameleon.html>). We also thank very much our colleagues who have been involved in these projects for their fruitful exchanges and discussions.

## References

1. Balme, L., Demeure, A., Barralon, N., Coutaz, J., Calvary, G.: CAMELEON-RT: a Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. In: Proc. of EUSAI'2004. Lecture Notes in Computer Science, Vol. 3295. Springer-Verlag, Berlin (2004) 291–302
2. Bouillon, L., Vanderdonckt, J., Chow, K.C.: Flexible Re-engineering of Web Sites; In: Proc. of 8<sup>th</sup> ACM Int. Conf. on Intelligent User Interfaces IUI'2004 (Funchal, 13-16 January 2004). ACM Press, New York (2004) 132–139
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, Vol. 15, No. 3 (June 2003) 289–308
4. Constantine, L.L.: Canonical Abstract Prototypes for Abstract Visual and Interaction Design. In: Proc. of 10<sup>th</sup> Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Funchal, June 11-13, 2003). Lecture Notes in Computer Science, Vol. 2844. Springer-Verlag, Berlin (2003) 1–15. Accessible at <http://www.foruse.com/articles/abstract.pdf>
5. Coutaz, J.: PAC, an Object Oriented Model for Dialog Design. In: Proc. of 2<sup>nd</sup> IFIP International Conference on Human-Computer Interaction Interact'87 (Stuttgart, September 1-4, 1987). North Holland, Amsterdam (1987) 431–436.
6. Coyette, A., Vanderdonckt, J.: A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. In: Proc. of 10<sup>th</sup> IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2005 (Rome, 12-16 September 2005). Lecture Notes in Computer Science. Springer-Verlag, Berlin (2005)

7. Florins, M., Vanderdonckt, J.: Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems. In: Proc. of 8<sup>th</sup> ACM Int. Conf. on Intelligent User Interfaces IUI'2004 (Funchal, 13-16 January 2004). ACM Press, New York (2004) 140–147
8. Furtado, E., Furtado, V., Soares Sousa, K., Vanderdonckt, J., Limbourg, Q.: KnowiXML: A Knowledge-Based System Generating Multiple Abstract User Interfaces in UsiXML. In: Proc. of 3<sup>rd</sup> Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004 (Prague, November 15-16, 2004). ACM Press, New York (2004) 121–128
9. Grolaux, D., Van Roy, P., Vanderdonckt, J.: Migratable User Interfaces: Beyond Migratory User Interfaces. In: Proc. of 1<sup>st</sup> IEEE-ACM Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services MOBIQUITOUS'04 (Boston, August 22-25, 2004). IEEE Computer Society Press, Los Alamitos (2004) 422–430
10. Jacob, R., Limbourg, Q., Vanderdonckt, J.: Computer-Aided Design of User Interfaces IV. Proc. of 5<sup>th</sup> Int. Conf. of Computer-Aided Design of User Interfaces CADUI'2004 (Funchal, 13-16 January 2004). Information Systems Series, Kluwer Academics, Dordrecht (2005)
11. Kolski, Ch., Vanderdonckt, J.: Computer-Aided Design of User Interfaces III. Proc. of 4<sup>th</sup> Int. Conf. of Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, 15-17 May 2002). Information Systems Series, Kluwer Academics, Dordrecht (2002)
12. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V.: UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In: Proc. of 9<sup>th</sup> IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11<sup>th</sup> Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Springer-Verlag, Berlin (2005)
13. Limbourg, Q., Multi-path Development of User Interfaces. Ph.D. thesis. Université catholique de Louvain, IAG-School of Management. Louvain-la-Neuve (Nov. 2004).
14. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley, New York (2004)
15. Molina, J.P., Vanderdonckt, J., Montero, F., Gonzalez, P.: Towards Virtualization of User Interfaces. In: Proc. of 10<sup>th</sup> ACM Int. Conf. on 3D Web Technology Web3D'2005 (Bangor, March 29-April 1, 2005). ACM Press, New York (2005)
16. Montero, F., Lozano, M., González, P.: IDEALXML: an Experience-Based Environment for User Interface Design and pattern manipulation. Technical Report DIAB-05-01-4. Universidad de Castilla-La Mancha, Albacete (2005).
17. Schlee, M., Vanderdonckt, J.: Generative Programming of Graphical User Interfaces. In: Proc. of 7<sup>th</sup> Int. Working Conference on Advanced Visual Interfaces AVI'2004 (Gallipoli, May 25-28, 2004). ACM Press, New York (2004) 403–406
18. Vanderdonckt, J., Bouillon, L., Chieu, K.C., Trevisan, D.: Model-based Design, Generation, and Evaluation of Virtual User Interfaces. In: Proc. of 9<sup>th</sup> ACM Int. Conf. on 3D Web Tech. Web3D'2004 (Monterey, April 5-8, 2004). ACM Press, New York (2004) 51–60
19. Vanderdonckt, J., Bodart, F.: Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In: Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 April 1993). ACM Press, New York (1993) 424–429
20. Vanderdonckt, J., Puerta, A.R.: Computer-Aided Design of User Interfaces II. Proc. of 3<sup>rd</sup> Int. Conf. of Computer-Aided Design of User Interfaces CADUI'99 (Louvain-la-Neuve, 21-23 October 1999). Information Systems Series, Kluwer Academics, Dordrecht (1999)
21. Vanderdonckt, J.: Computer-Aided Design of User Interfaces. Proc. of 2<sup>nd</sup> Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96 (Namur, 5-7 June 1996). Collection « Travaux de l'Institut d'Informatique » n°15. Presses Universitaires de Namur, Namur (1996)

# Toward Semantic Interoperability of Heterogeneous Biological Data Sources

Sudha Ram

Eller Professor of MIS, Director, Advanced Database Research Group,  
Eller College of Management, University of Arizona,  
Tucson, AZ 85721  
ram@eller.arizona.edu  
<http://vishnu.eller.arizona.edu/ram>

Genomic researchers use a number of heterogeneous data sources including nucleotides, protein sequences, 3-D Protein structures, taxonomies, and research publications such as MEDLINE. This research aims to discover as much biological knowledge as possible about the properties and functions of the structures such as DNA sequences and protein structures and to explore the connections among all the data, so that the knowledge can be used to improve human lives. Currently it is very difficult to connect all of these data sources seamlessly unless all the data is transformed into a common format with an id connecting all of them. The state-of-the-art facilities for searching these data sources provide interfaces through which scientists can access multiple databases. Most of these searches are primarily text-based, requiring users to specify keywords using which the systems search through each individual data source and returns results. The user is then required to create the connections between the results from each source. This is a major problem because researchers do not always know how to create these connections. To solve this problem we propose a semantics-based mechanism for automatically linking and connecting the various data sources. Our approach is based on a model that explicitly captures the semantics of the heterogeneous data sources and makes them available for searching. In this talk I will discuss issues related to capturing the semantics of biological data and using these semantics to automate the integration of diverse heterogeneous sources.

# The Association Construct in Conceptual Modelling – An Analysis Using the Bunge Ontological Model

Joerg Evermann

School of Information Management, Victoria University,  
Wellington, New Zealand

Joerg.Evermann@mcs.vuw.ac.nz

**Abstract.** Associations are a widely used construct of object-oriented languages. However, the meaning of associations for conceptual modelling of application domains remains unclear. This paper employs ontological analysis to first examine the software semantics of the association construct, and shows that they cannot be transferred to conceptual modelling. The paper then explores associations as 'semantic connections' between objects and shows that this meaning cannot be transferred to conceptual modelling either.

As an alternative to the use of associations, the paper proposes using shared properties, a construct that is rooted directly in ontology. An example from a case study demonstrates how this is applied. The paper then shows an efficient implementation in object-oriented programming languages to maintain seamless transitions between analysis, design, and implementation.

## 1 Introduction

Object-oriented modelling languages are increasingly being used for describing business and organizational application domains (conceptual modelling). In order to have well-defined meaning, their constructs must be defined in terms of the elements of the application domain [1]. The use of constructs without clearly defined meaning can lead to ambiguous or confusing models.

However, the meaning of the association construct remains unclear, as the following attempts at a definition show<sup>1</sup>:

An association is "the simplest form of a relationship" [2, p. 195].

"An association represents the relationships between objects and classes" [3, p. 26].

"Relationships associate one object with another" [4, p. 18].

---

<sup>1</sup> Note that this concerns the semantics for conceptual modelling only. Software semantics for associations are discussed in Sect. 3.

"If two classes have an association between them, then instances of these classes are, or might be, linked." [2].

"An association sets up a connection. The connection is some type of fact that we want to note in our model" [5, p. 105].

The original definition by Rumbaugh sheds little light on the issue:

"A relation associates objects from  $n$  classes. ... A relation is an abstraction stating that objects from certain classes are associated in some way." [6, p. 466].

This lack of clarity remains even in the latest UML standard:

"An association defines a semantic relationship between classifiers." [7, p. 2-19].

To provide a clear definition of associations in terms of the elements of the application domain, we must first determine what exists, or is assumed to exist, in the application domain. Ontologies specify what concepts exist in a domain and how they are related. Hence, to define the meaning of an association, we must map them to an ontological concept [1, 8]. This mapping must support the syntactic features of associations as much as possible. For example, associations connect two or more classes of objects. Hence, they should be mapped to an ontological concept that connects two or more sets of things or objects.

Note that this paper is concerned with association semantics from the perspective of an application domain analyst, not a software designer or programmer. In fact, association semantics are problematic also for the latter case, as shown in [9, 10]: The relationship between the association construct and programming language implementations is often unclear.

The paper proceeds as follows. Section 2 introduces the ontology that is adopted for this analysis. Next, the paper identifies the usage and the semantics of associations in software modelling (Sect. 3). It shows that software semantics cannot be transferred to conceptual modelling. Section 4 examines associations as 'semantic connections' and shows that there is no ontological concept with similar use or meaning. Hence, associations have no semantics when used for conceptual modelling.

The use of mutual shared properties is proposed as an alternative to the use of associations in conceptual modelling. We advocate the notation of association class attributes to represent mutual properties (Sect. 5). An example from a case study demonstrates this technique using a real modelling situation (Sect. 6). Finally, Sect. 7 demonstrates that the proposed technique can be efficiently and transparently implemented in object-oriented programming languages in order to maintain seamless transitions between object-oriented system analysis, software design, and implementation.

## 2 Ontology

This research does not relate to a particular application domain but to a language construct that is not domain-specific. Hence, an ontology on a suitable level of abstraction is required. A number of proposed high-level or upper-level ontologies exist [11, 12, 13, 14, 15, 16, 17].

Among these, the ontology proposed by Bunge [12] stands out because it has been empirically validated in a variety of applications and application domains [18, 19, 20]. Furthermore, it has repeatedly been shown to provide a good benchmark for the analysis of modelling languages and language constructs [21, 22, 23, 8, 24].

Bunge [12] proposes that the world is made up of... which physically exist in the world. A thing possesses... each of which corresponds to a... For example, being colored red is an individual property of a particular thing; color is a property in general.

Properties are either intrinsic or mutual... are ones that a thing possess by itself, e.g. color, whereas... are shared between two or more things, e.g. voltage of a processor and memory unit, temperature of a heater and surrounding air, etc.

Two or more things can interact with each other... is defined by the history of a thing: If the way in which the properties of a thing change depends on the existence of another thing, then the second is said to act on the first. Ontologically, for things to interact, e.g. for a thing  $A$  to act on a thing  $B$ , there must exist a mutual property  $P$  of  $A$  and  $B$ . A change of  $P$  in  $A$  is also a change of  $P$  in  $B$ . The change of  $P$  may then cause further changes in  $B$ . For example, for a heater to warm the air in a room, the heater and surrounding air share a mutual property, temperature. When the action of the heater changes the temperature, this change leads to changes in room temperature.

## 3 Software Semantics of Associations

Associations, introduced to graphical object-oriented modelling in [6], originate in object-oriented programming languages. Here, they have well defined meaning: "Associations are often implemented in programming languages as pointers from one object to another" [25, p. 27], "a relation expresses associations often represented in a programming language as pointers from one object to another." [6, p. 466]. For example, the association "attends" in Fig. 1 may be implemented in the following way, making use of pointers. These pointers are then used to enable method calls.

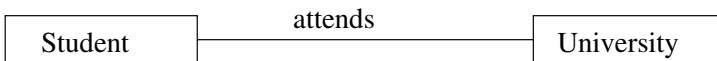


Fig. 1. Association example

```

class University {
    void PayFees(int studentno, int semester, float amount);
    ... }
class Student {
    AttendedUniversity *University;
    ... }
...
AttendedUniversity -> PayFees(12345, 2, 880.00);

```

The above example shows the software semantics of associations: They provide the means for interaction by message-passing in software<sup>2</sup>. Booch [26] terms this a *use association*, Coad & Yourdon [27] call them *use relationships*. For the remainder of the paper we call them "use associations"<sup>3</sup>.

Associations as enablers of message-passing are useful for describing software. However, the following arguments show that these software semantics are not transferable to the conceptual modelling of application domains.

First, the interaction mechanism in [12] is not based on message passing. Ontologically, for things to interact, e.g. for a thing<sup>4</sup>  $A$  to act on a thing  $B$ , there must exist a mutual property  $P$  of  $A$  and  $B$ . A change of  $P$  in  $A$  is also a change of  $P$  in  $B$ . The change of  $P$  may then cause further changes in  $B$ .

Second, the message passing mechanism for interaction has previously been examined with respect to Bunge's ontology and found to be an unsuitable construct or mechanism for describing real world business domains [8, 23, 29]<sup>5</sup>. Consider the following examples:

- The machine sends a message to the part to move itself to a new location
- The general ledger sends a message to an office desk to depreciate its value
- A truck sends a message to the crate to load itself onto the loading dock

---

<sup>2</sup> Although, as pointed out in [9], UML is contradictory in allowing a dependency to express dynamic behaviour on the classifier level, while requiring a link, an association instance, to enable dynamic behaviour on the instance level. In this paper we assume that, as the instance level requires a link, the classifier level must require an association to enable dynamic behaviour.

<sup>3</sup> We note that there exists a `<< use >>` stereotype of a dependency in UML. However, following the argument by [9] in the previous note, we subsume this notion under our "use association". [9] further differentiates between the static pointer aspect and the dynamic message-passing aspect. In her terms, we subsume both aspects under "use association".

<sup>4</sup> When the term "thing" is used, the discussion relates to the ontology in [12]. When the term "object" is used, the discussion relates to object-oriented languages. Things in the application domain are represented by objects in an object-oriented description [28, 22].

<sup>5</sup> This also agrees with the informal assessment in [30].



Such descriptions are common in software specifications but such messages have not been observed between these things in the real world. Clearly, a machine does not send messages to parts. Instead the parts are moved by an operator<sup>6</sup>.

Third, messages should not be interpreted as ontological things. Interaction with such message things would necessarily also have to occur by messages. Hence, a thing *A* interacts with thing *B* by exchanging the message thing  $M_1$ . For this to occur, *A* must interact with thing  $M_1$  by means of another message thing  $M_2$ , etc. This leads to an infinite regress<sup>7</sup>.

In summary, as message-passing does not have an ontological equivalent, "use associations" cannot be mapped to a suitable ontological concept. Thus, according to [1, 8] they possess no ontological semantics for the modelling of application domains.

## 4 Connection Semantics of Associations

Rumbaugh et al. [25] maintain the importance of an association as a conceptual, real-world construct: "We nevertheless emphasize that associations are a useful modelling construct for ... real-world systems ..." [25, p. 31], "It is important that relations be considered a semantic construct" [6, p. 467], "associations define the way objects of various types can be linked or connected - enabling the construction of conceptual networks" [32, p. 259], a "class relationship might indicate some kind of semantic connection" [26, p. 96]<sup>8</sup>.

We claim that associations used as "connections" between objects ("connection associations") have no ontological equivalent for the following reason. The term "semantic", as used in [6, 26], implies meaning and human interpretation. Hence, semantic connections are imposed on a domain as perceived by an observer, rather than directly observable in the domain. They represent properties that are

For example, the fact that a student attends a university (Fig. 1) is not observable in the domain; only the properties (e.g. student number, fee balance), and the behaviour of the student (e.g. attending class, sitting exams) and the university are observable. Some behaviour may be relevant and is interpreted as the student attending the university. To other observers, or for different model purposes, this behaviour may be irrelevant or may be interpreted as a different semantic connection.

<sup>6</sup> Note that it is quite possible in organizational settings for human actors to pass messages to each other: Letters can be exchanged, invoices sent and orders received. In contrast to the messages between parts and machines, ledgers and desks, or trucks and crates, the letters, invoices, and orders that are exchanged between human actors are substantial, physical things. .

<sup>7</sup> A similar argument is used in [31] to argue against association instances as objects.

<sup>8</sup> In Stevens' analysis [9], associations as connections correspond roughly to "static associations", although the latter are defined by their relationship to implementation, rather than their relationship to application domain elements.

Examining the way in which associations are used, e.g. the "attends" association in the above example, shows that there exist two distinct types of semantic associations.

First, some associations represent functions of past interaction of objects. Consider the association 'enrolled' between a student and a university. 'Being enrolled' is a result of past or ongoing interaction, namely that of the registration and enrollment process. The definition in terms of interaction also shows that the association is viewer or modeller dependent: A different definition of 'enrolled' may be based on class attendance rather than the act of registration. The association 'being on' between a shipping crate and the loading dock also depends on interaction, but not between the crate and the dock. These never interacted directly. Instead, this association is the result of past interaction between e.g. the crate and the forklift.

Second, consider the association 'distance' between two objects. Distance is defined by an observer or modeller based on properties of things, such as their location. Consequently, different distance measures are possible, for example in terms of road distance, traveling time, etc. This kind of semantic connection between objects does not depend on interaction, but represents functions of individual properties of things.

We conclude that, as semantic associations are observer dependent functions either of interaction or of intrinsic properties, they do not correspond to anything that exists in the application domain. Hence, they should be explicitly represented in functional form, rather than by an association construct that, because of its programming heritage, obscures their nature.

## 5 Conceptual Modelling with Mutual Properties

The previous sections showed a lack of ontological semantics for associations as 'use associations' (Sect. 3) and 'connection associations' (Sect. 4). Hence, they should not be used for describing application domains. Instead, we propose using ontological concepts directly. Since 'use associations' are intended to represent interaction, and 'connection associations' represent functions of the interaction history or individual properties, the relevant ontological concepts are those related to interaction and properties.

The adopted ontology [12] specifies that two or more things may share mutual properties. Hence, any change of a mutual property in one thing is a change in all the things that share the property (Sect. 2). When a series of changes in an object  $A$  involves changes to a mutual property  $P$ , this may start a series of changes in the things that share this property, e.g. thing  $B$ . Thing  $A$  has acted on thing  $B$  through property  $P$ .

Hence, interaction can be described in terms of mutual properties. Instead of employing 'use associations' with poorly defined ontological semantics, we propose using mutual properties for conceptual modelling.

Using mutual properties in conceptual models requires a language construct (graphical symbol) to represent them. For this, we use attributes of association

classes: (1) Intuitively, the idea of an attribute corresponds well to the ontological concept of a property [22, 28]. (2) Attributes of association classes are graphically shown as connecting two or more objects (e.g. Fig. 2). Intuitively, this corresponds well with the idea of a single property being shared by two or more things.

Note that we merely borrow, for sake of convenience and familiarity, the graphical notation of association class attributes from UML. Associations and association classes themselves have no ontological interpretation and should not be used for conceptual modelling, as argued in Secs. 3 and 4. However, UML requires the use of a class symbol to represent attributes. This is a necessary evil that we accept in order to avoid introducing a new notation element. For this reason, association class symbols contain no name in the figures in Sec. 6. An alternative would be to introduce a new graphical or textual notation for shared attributes.

Ontologically, the history of interaction is the history of changes to mutual properties (Sect. 4). No graphical modelling exists in common object-oriented languages that could be used to express such functions. Instead, we propose to use simple textual notation, e.g. the following example in Prolog like notation:

```
property(downtown, location, 10).
property(campus, location, 20).
property(hospital, location, 15).
...
distance(O1, O2, D) :-
    property(O1, location, X),
    property(O2, location, Y),
    D is X - Y.
```

Now we can ask for the distance from downtown to the campus:

```
distance(downtown, campus, D).
```

Similarly, an example for a function of the event history of the world is the following employment association (again in Prolog):

```
history(acmecorp, johnsmith, hires, 20030701).
history(acmecorp, janedoe, hires, 20031001).
history(acmecorp, johnsmith, fires, 20040101).
...
connection(O1, O2, employs) :-
    history(O1, O2, hires, Time1),
    \+ history(O1, O2, fires, Time2).
connection(O1, O2, employs) :-
    history(O1, O2, hires, Time1),
    history(O1, O2, fires, Time2),
    Time1 > Time2.
```

We can now ask whether Acme Corp. employs Jane Doe:

```
connection(acmecorp, janedoe, employs).
```

As noted above (Sect. 4), semantic associations are relative to a modeller or observer. Hence, they should not be a part of the domain model, but explicitly noted as part of a perspective on or interpretation of the domain. This also means that different domain observers, modellers, or users of the final information system, can define a different set of functions that are relevant to them.

## 6 Case Study Example

This section presents an example that demonstrates the proposed conceptual modelling technique. The technique was used in an actual IS development project, carried out at a large North American university. The project goal was to develop an Internet based system to allow prospective students the opportunity to update their application information on an ongoing basis and enable them to see whether they meet application criteria.

This section focuses on the identification and representation of mutual properties and interaction. Only a brief excerpt of the complete analysis is provided, focusing on high school students, high schools, teachers, the university and the application process.

High school students and teachers are modelled as object classes. The relevant interaction (for purposes of the analysis) between a student and a high school is that of receiving course grades. Interaction occurs by means of changes to mutual properties (Sect. 5). Hence this is modelled using attributes of an association class that represent the mutual properties. In Fig. 2 the attributes are multi-valued, i.e. there is a course name, a course year and a course grade for each course the student completed<sup>9</sup>. Teachers interact with students by changing the values of these attributes<sup>10</sup>.

The properties in Fig. 2 arose themselves out of interaction. Therefore, prior mutual properties between teachers and students must have existed, such as homework submitted and read. Ultimately, the mutual properties are those of an interaction medium that is manipulated by the interacting objects. However, such media are rarely of interest to the conceptual modeller and are thus abstracted from. Note that this abstraction is a conscious decision of the modeller, and is always dependent on the purpose of the model.

Communication between the high school and the student can lead to meaningful semantic connections. For example, one can model the semantic connection

---

<sup>9</sup> Note again that we borrow only the graphical notation of association class attributes from UML. Associations and association classes themselves have no ontological interpretation. However, UML requires the use of a class symbol to represent attributes. Note that the association class itself is not named, as we have not assigned it ontological meaning (Sect. 5).

<sup>10</sup> Obviously, there occurs more interaction, e.g. in the classroom, which is not relevant to the university's admission system.

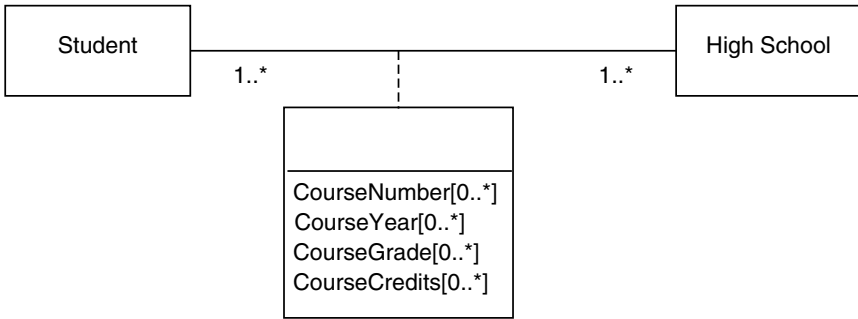


Fig. 2. Example: Student - High School interaction

'has graduated from' as a function of the interaction history: When the sum of course credits is above a certain threshold, we consider the student to have graduated<sup>11</sup>. Having graduated in this sense is an observer- or modeller-dependent function, rather than an object or event in the application domain. It may be of interest only in certain contexts, or it may be defined differently by different observers or modellers, e.g.

```

history(janeDoe, centralHigh, 100, 20040701).
history(johnSmith, southernHigh, 120, 20040701).
history(jimMiller, northernHigh, 150, 20041201).
...
property(Student, HighSchool, graduated) :-
    history(Student, HighSchool, creditEarned, Time1),
    creditEarned > 100.
  
```

We can now ask whether Jane Doe has graduated from Central High:

```
property("janeDoe, centralHigh, graduated).
```

Changing the mutual properties in a certain way will lead to a change in the student where she considers applying to a university. Applying to the university is interaction. Interaction implies mutual properties between the student and the university that can be manipulated by the student. We can either abstract from this information and simply call it "application information", or we can model the specific properties, e.g. "program applied for", "school grades submitted", etc. The student can change these shared properties. As a result of these changes, the admission process in the university is initiated (Fig. 3)<sup>12</sup>.

This type of modelling forces the modeller to make a distinction between the grades awarded by the high school (they are shared between the school and the student), and the grades reported by the student for the admission request (they

<sup>11</sup> Abstracting from the formalities which are often attached for graduating.

<sup>12</sup> See previous note.

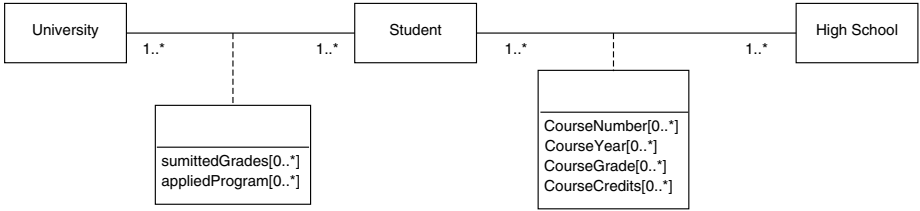


Fig. 3. Example: Student - University interaction

are shared between the student and the university). It may well be that these are different, e.g. due to the fact that the school may revise preliminary grades, or the student reports only a subset of grades to the university.

Changes to the shared properties are interpreted as interactions. Consequently, they can be modelled in UML interaction diagrams. Fig. 4 shows an example corresponding to the above excerpt of the case study.

A subsequent discussion with the project leader showed that modelling of mutual properties can help explicate the meaning of association class attributes. It forces the developer to identify precisely what is represented: "It's normally difficult to model a course ..., because it is a relationship. ... What do you mean by a course? The curriculum, the interaction, the grade?". In contrast, the above model (Fig. 2), with the clear ontological semantics proposed in Sect. 5, shows that students and universities share a set of properties that can be modified by either to initiate interaction. The lead system designer also noted the beneficial effect of the clear semantics for association class attributes: "Visually, this ... gives you a better sense of the relationships."

The case study results show that the proposed modelling method is feasible and can be applied in real projects. The subsequent discussion shows that the technique leads to clearer models and to models with clearer meaning than the use of associations for conceptual modelling.

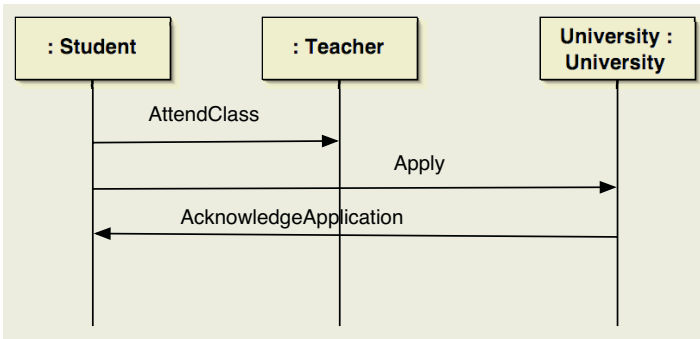


Fig. 4. Example: Student - University sequence diagram

## 7 Software Design and Implementation Without Associations

The main advantage of object-oriented methodologies is the seamless transition from conceptual modelling to system design, and to system implementation using the same paradigm and set of language constructs. This section shows how the proposed conceptual modelling technique can be seamlessly transferred to system implementation.

We describe a technique for object-oriented implementation that does not use object references or method calling. It implements shared mutual properties (conceptual level) by 'binding' object attributes (programming level) of two or more objects to ensure that they maintain identical values.

Attribute binding can be achieved in an efficient and completely transparent way by using aspect-oriented programming (AOP) techniques<sup>13</sup>. For demonstration purposes, an aspect for attribute binding has been developed in Aspect/J, a widely used AOP extension to Java [33]. The aspect allows binding together any two or more attributes of different objects. The following code shows two fictitious objects *a1* and *a2*, both instances of class *A*, that share a property single mutual property by binding their respective properties *varA* of *a1* and *varB* of *a2*. The example demonstrates that this binding is transparent to the application developer: No special bound properties or accessor methods need to be declared.

```
public class A {
    public Integer varA, varB;
}

public static void main() {
    A a1 = new A();
    A a2 = new A();

    PropBinding.addBindings(a1, "varA", a2, "varB", "InteractionName");
}
```

As can be seen from the example, an interaction name can be associated with each binding. The aspect monitors the write accesses to bound attributes and updates them accordingly, ensuring that they maintain identical values. The aspect also maintains a history of every update of bound attributes. This history can be accessed and searched by object identifier and interaction name so that functions of the interaction history (as described in Sect. 5) can be defined.

---

<sup>13</sup> AOP allows the separate development and implementation of different aspects of an implementation (e.g. logging, security, persistence) in a transparent way. Individually programmed aspects are woven together when compiling the final software product. Mature AOP tools are available for most languages (e.g. AspectWerkz, Aspect/J, AspectC++, Aspect#).

Since two or more bound attributes represent a single shared mutual property, changes to one attribute are propagated to all bound attributes in a single atomic step. The aspect recursively propagates changes along bound attributes, until all bound attributes possess equal values. Only then is the program allowed to resume execution.

As there exist no object references, objects cannot interact by method calls along such references. Consequently, in a single-threaded model, explicit notification is necessary to pass control from one object to another. In a multi-threaded model, notification is not necessary, as control is not passed between objects, but may be desirable in certain situations or for certain applications.

**First, the order of callback execution.** In a single-threaded model, the implementation of the aspect allows explicit registration of a callback method. This presents two potential problems. First, the order of callback execution must be determined. In the current implementation, callbacks are executed in the order in which the attribute bindings are declared; other execution orderings are possible.

Second, a set of objects may possess shared mutual properties in such a configuration that actions by a notified object change bound attribute values before all remaining objects have received notification of the original changes. For example, an object  $t$  changes the value of a bound attribute  $k$  from value  $a$  to  $b$ . After propagating this change to objects  $u$  and  $v$ , the notification callback of object  $u$  is called and changes the value of the attribute  $k$  from  $b$  to  $c$  before object  $v$  is notified of the change from  $a$  to  $b$  by calling its callback method. From the perspective of the object  $v$ , the first change to  $b$  never happened as the object never gained control while  $k$  possessed value  $b$ . Note that objects cannot be notified before a change has been propagated to all bound attributes. This is because the attributes represent a single shared mutual property, and thus must be updated in a single atomic action.

Therefore, in the single-threaded model, the semantics of the implementation depend on the ordering of execution of callback. This requires great care by the programmer.

**Second, the order of notification.** In multi-threaded applications, no callback methods are possible. Instead, all notification must be done by means of event signaling. The implemented aspect provides an event queue for every thread/object into which notifications can be added. Consequently, in the multi-threaded model the semantics of the attribute binding are well-defined and independent of the ordering of callback execution.

In this model, too, changes to attributes can occur before prior changes have been processed by the object. In the multi-threaded model, each object possesses its own notification queue, containing notifications about attribute changes. While a change notification for a bound attribute is still queued, i.e. it has not been processed by the object yet, this attribute may be changed again. However, for the same reasons as in the single-threaded model, only net change notifications are provided. Hence, whenever a new change notification is added



to the queue, the net-effect of this and all previously queued notifications will be computed. This net-effect notification replaces all other elements in the queue that notify of changes in the same attribute.

In summary, the implementation of mutual properties by attribute binding is efficient (linear in the number of bindings between properties), and possesses well-defined execution semantics.

As an alternative to the aspect-oriented implementation of mutual properties presented above, an existing programming library has been considered. Property binding has been proposed and implemented for user-interface objects in a C++ library (LibPropC++) [34]. However, compared to the previous approach, this library has a number of weaknesses. (1) Its use is not transparent as it requires that object attributes must be explicitly declared as properties and appropriate accessor methods must be provided. (2) Binding of attributes in LibPropC++ is not designed to replace method calls. Objects still possess object reference pointers and need to call methods of other objects. (3) The library does not maintain an interaction history, so it cannot provide a foundation on which functions of past interaction can be defined.

## 8 Conclusions and Further Research

This research was motivated by problems with the meaning of associations in conceptual models. We identified the semantics of associations with respect to software and attempted to transfer this to conceptual modelling. However, as the interaction mechanisms in application domains (specified by an ontology) do not rely on message passing and method calling, associations as enablers of message passing have no application domain meaning.

We then discussed the intended use of associations to indicate semantic "connections" between objects. Our analysis showed that these "connections" are used to describe observer dependent properties, rather than substantial elements in the application domain. Hence, associations as "connections" have no ontological semantics in conceptual modelling.

This paper proposes an alternative technique for conceptual modelling that is rooted directly in ontology. To this effect, it defines ontological semantics for attributes of association classes, by mapping them to mutual properties. These mutual properties are the linkage between things and the means by which interaction occurs.

A case study was presented that demonstrates the use of this modelling technique. The paper further demonstrated that appropriate software technologies exist to seamlessly transfer this ontologically based modelling technique to software design and implementation.

To summarize, since the meaning of associations is undefined, we suggest not to use them. Instead, we propose using ontological concepts directly, by representing them as association class attributes. The contributions of this paper are threefold. (1) We have identified and pointed out ambiguities in the meaning of

associations. (2) We have shown that associations, as intended by their originators, have no ontological equivalent, i.e. no semantics. (3) We then proposed an ontologically based modelling technique and shown that it can be implemented efficiently in object-oriented programming languages.

While the initial case study, described in parts in Sect. 6, shows that the modelling and implementation approach are feasible, further research in three areas is needed. (1) The implementation on the programming level must be further analyzed to firmly define the implementation semantics of the approach. (2) The approach needs to be evaluated in a wider set of domains. To this effect, further case study applications will be undertaken. (3) Finally, the benefits of the proposal, in terms of model interpretation and model understanding, need to be determined in a controlled setting.

## References

1. Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"? *IEEE Computer* (2004) 64–72
2. IBM: Developing object-oriented software: an experience-based approach. Prentice Hall, Inc., Upper Saddle River, NJ (1997)
3. Bahrami, A.: Object oriented systems development. Irwin/McGraw-Hill, Boston, MA (1999)
4. Embley, D.W.: Object-oriented systems analysis: a model-driven approach. Prentice Hall, Inc., Englewood Cliffs, NJ (1992)
5. Siegfried, S.: Understanding object-oriented software engineering. IEEE Press, New York, NY (1995)
6. Rumbaugh, J.: Relations as semantic constructs in an object-oriented language. In: Proceedings of the 1987 Conference on Object Oriented Programming Systems and Languages and Applications, Orlando, FL., ACM Press (1987) 466–481
7. OMG: The Unified Modelling Language Specification. Version 1.5. (2003)
8. Wand, Y., Weber, R.: On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems* (1993) 217–237
9. Stevens, P.: On the interpretation of binary associations with the unified modelling language. *Software and Systems Modelling* **1** (2002) 68–79
10. Genova, G., Llorens, J., Martinez, P.: The meaning of multiplicity of n-ary associations in UML. *Software and Systems Modelling* **1** (2002) 86–97
11. Bennett, B.: Space, time, matter and things. In: Proceedings of the 2001 International Conference on Formal Ontologies in Information Systems FOIS, Ogunquit, Maine. (2001) 105–116
12. Bunge, M.A.: *Ontology I: The Furniture of the World*. Volume 3 of *Treatise On Basic Philosophy*. D. Reidel Publishing Company, Dordrecht, Holland (1977)
13. Chisholm, R.: *A Realistic Theory of Categories - An Essay on Ontology*. Cambridge University Press, Cambridge (1996)
14. Degen, W., Heller, B., Herre, H., Smith, B.: GOL: A general ontological language. In: Proceedings of the 2001 Conference on Formal Ontologies in Information Systems FOIS, Ogunquit, MA. (2001) 34–46
15. Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F.: OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems* (2001) 38–45

16. Niles, I., Pease, A.: Towards a standard upper ontology. In: Proceedings of the 2nd International Conference on Formal Ontologies in Information Systems FOIS, Ogunquit, Maine 2001. (2001) 2–9
17. Sowa, J.F.: Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks Cole, Pacific Grove, CA (2000)
18. Bodart, F., Patel, A., Sim, M., Weber, R.: Should optional properties be used in conceptual modelling? A theory and three empirical tests. *Information Systems Research* **12** (2001) 384–405
19. Evermann, J.: Using Design Languages for Conceptual Modelling: The UML Case. PhD thesis, University of British Columbia, Canada (2003)
20. Gemino, A.: Empirical Comparisons of Systems Analysis Modeling Techniques. PhD thesis, University of British Columbia, Canada (1999)
21. Green, P., Rosemann, M.: Integrated process modelling: An ontological analysis. *Information Systems* **25** (2000) 73–87
22. Opdahl, A., Henderson-Sellers, B.: Ontological evaluation of the UML using the Bunge-Wand-Weber model. *Software and Systems Modeling* **1** (2002) 43–67
23. Parsons, J., Wand, Y.: Using objects for systems analysis. *Communications of the ACM* **40** (1997) 104–110
24. Wand, Y., Storey, V.C., Weber, R.: An ontological analysis of the relationship construct in conceptual modeling. *ACM Transactions on Database Systems* **24** (1999) 494–528
25. Rumbaugh, J., et al.: Object Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, NJ (1991)
26. Booch, G.: Object oriented design with applications. Benjamin/Cummings, Redwood City, CA (1991)
27. Coad, P., Yourdon, E.: Object-Oriented Analysis. Yourdon Press, Englewood Cliffs, NJ (1990)
28. Evermann, J., Wand, Y.: An ontological examination of object interaction in conceptual modeling. In: Proceedings of the Workshop on Information Technologies and Systems WITS'01, New Orleans, December 15-16, 2001. (2001) 91–96
29. Parsons, J., Wand, Y.: The object paradigm – two for the price of one? In: Proceedings of the Workshop on Information Technology and Systems WITS 1991, New York, NY. (1991) 308–319
30. Cook, S., Daniels, J.: Designing object systems: object-oriented modelling with Syntropy. Prentice Hall, Hertfordshire, UK (1994)
31. Graham, I., Bischoff, J., Henderson-Sellers, B.: Associations considered a bad thing. *Journal of Object-Oriented Programming* **9** (1997) 41–48
32. Martin, J., Odell, J.J.: Object-oriented analysis and design. Prentice Hall, Englewood Cliffs, NJ (1992)
33. Laddad, R.: AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications, Greenwich, UK (2003)
34. Porton, V.: Binding together properties of objects. <http://ex-code.com/articles/binding-properties.html> (2004) Last accessed Sept 23, 2004.

# Computing the Relevant Instances That May Violate an OCL Constraint

Jordi Cabot<sup>1,2</sup> and Ernest Teniente<sup>2</sup>

<sup>1</sup> Estudis d'Informàtica i Multimèdia, Universitat Oberta de Catalunya  
jcabot@uoc.edu

<sup>2</sup> Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya  
teniente@lsi.upc.edu

**Abstract.** Integrity checking is aimed at efficiently determining whether the state of the information base is consistent after the application of a set of structural events. One possible way to achieve efficiency is to consider only the relevant instances that may violate an integrity constraint instead of the whole population of the information base. This is the approach we follow in this paper to automatically check the integrity constraints defined in a UML conceptual schema. Since the method we propose uses only the standard elements of the conceptual schema to process the constraints, its efficiency improvement can benefit any implementation of the schema regardless the technology used.

## 1 Introduction

A conceptual schema (CS) must include the definition of all relevant integrity constraints (ICs) [6] since they state conditions that each state of the information base (IB) must satisfy.

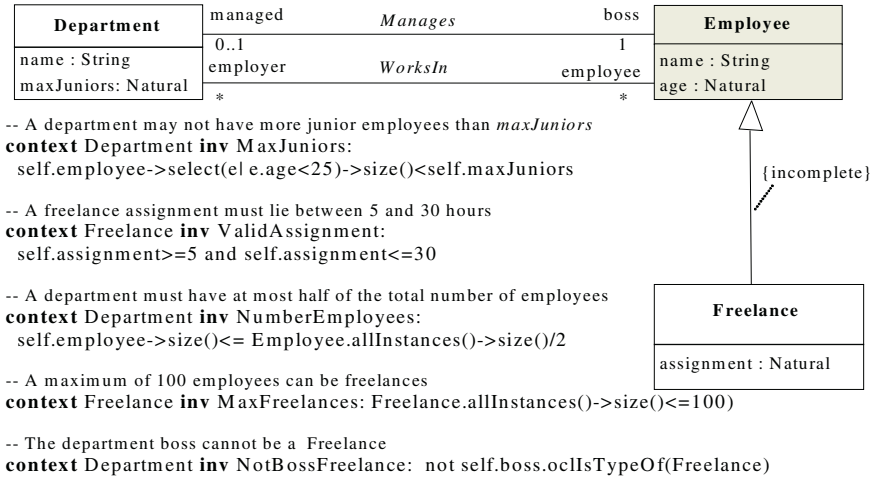
The content of the IB changes due to the execution of operations. In general, the effect of an operation over the IB may be specified by means of a set of structural events (see for instance [7], [11]). A structural event is an elementary change in the population of an entity type or relationship type such as: create object, delete object, update attribute, create link, etc.

The state of the IB resulting from the execution of an operation must be consistent with regards to the set of ICs specified over the CS. The traditional approach to deal with this problem is to reject those operations whose application would lead to an inconsistent state of the IB. This approach is usually known as integrity constraint checking and it requires verifying efficiently that the IB state obtained as a consequence of an operation execution does not violate any integrity constraint. For the sake of simplicity, we assume, without loss of generality, that each operation constitutes a single transaction and use both terms indistinctly.

In this paper we propose a new method to improve efficiency of integrity constraint checking in CSs specified in the UML [10] where constraints are written in the OCL [9]. Note that, as shown in [4], OCL can also be used to represent the graphic constraints expressed in the UML diagrams.

Constraints in OCL are defined in the context of a specific entity type, the *context entity type* (CET), and must be satisfied by all instances of that entity type. However, when verifying an IC not all instances must be taken into account since, assuming as usual that the IB is consistent before the update, only those that have been modified by the set of structural events applied over the IB may violate the IC.

Consider the running example of Fig. 1 to illustrate these ideas. After executing an operation that hires a junior employee (i.e. an employee under 25), not all departments must be taken into account to verify the constraints *MaxJuniors* and *NumberEmployees*. In fact, since the IB is assumed to be consistent before the operation execution, only the department where the employee starts working in may induce a violation of one of those integrity constraints.



**Fig. 1.** Example of a conceptual schema

Given a conceptual schema CS with a set of integrity constraints IC, our method generates a CS' with additional entity types, required to record the structural events issued by the operation, and where the definition of the original ICs has been modified to be able to verify them only in terms of the relevant instances. Moreover, the way we compute the relevant instances ensures that a constraint will not be verified if no structural event that may violate it has been issued by the operation.

In addition to the efficiency improvement, the main advantage of our method is that it works at the conceptual level, i.e. it is technology-independent, since the result of our method is a standard conceptual schema. Then, any architecture able to treat a CS to generate automatically its implementation can benefit from our method, no matter the target technology platform it generates. Pre-processing the original CS with our method allows any code-generation architecture to automatically generate efficient integrity constraints that are verified only in terms of its relevant instances.

The problem of efficient integrity checking has been widely addressed. However, as far as we know, ours is the first proposal to cope with this issue at a specification

level. Previous work addressing similar problems is always particular for a given technology. The best approaches are proposed in the fields of deductive [5] or relational [3] databases.

The work presented here extends our previous work in [1] where we proposed a method to compute the structural events that may violate an integrity constraint. However, that work did not care about how to check integrity constraints efficiently when one such structural event was issued by a transaction. This is precisely the main concern of this paper.

The paper is organized as follows. Section 2 introduces some basic concepts. Section 3 classifies the different kind of constraints according to the efficiency level our method can provide. In particular, our method improves the efficiency of instance constraints (section 4) and partial instance constraints (section 5). Finally, section 6 presents some conclusions and further work.

## 2 Determining the Structural Events That May Violate an IC

The first thing we need to take into account when computing the relevant instances of an integrity constraint is to determine the set of structural events that may cause its violation, i.e. its set of potentially violating structural events (PSE).

To compute the set of PSEs we consider the following kinds of structural events:

- InsertET(ET): insertion over the entity type *ET*.
- UpdateAttribute(Attr,ET): it updates the value of the attribute *Attr*.
- DeleteET(ET): it deletes an instance of the entity type *ET*.
- SpecializeET(ET): it specializes an instance of a supertype of the entity type *ET* to *ET*.
- GeneralizeET(ET): it generalizes an instance of a subtype of *ET* to *ET*.
- InsertRT(RT): creation of a new link in the relationship type *RT*.
- DeleteRT(RT): it deletes a link of the relationship type *RT*.

For instance, the event *InsertET(Freelance)* is a PSE for *ValidAssignment* since the new freelance may have an assignment below 5 or over 30, and thus, it may violate the constraint. On the contrary, the event *DeleteET(Freelance)* is not a PSE for that constraint since it may never induce a violation of *ValidAssignment*.

To compute the PSEs and the relevant instances that may violate an integrity constraint we assume that OCL constraints are represented as an instance of the OCL metamodel [9, ch. 8]. Therefore, we treat the OCL expression of the constraint as a binary tree where each node represents an atomic subset of the OCL expression (an operation, an access to an attribute or an association, etc.).

The root of the tree is the most external operation of the OCL expression. The left child of a node is the source of the node (the part of the OCL expression previous to the node). The right child of a node is the argument of the operation (if any). As an example, Fig. 2.1 shows the constraint *MaxJuniors* (*self.employee->select(e| e.age<25)->size(<self.maxJuniors)*) as an instance of the OCL metamodel. The operator '<' is the root of the tree. The left child is the source of the operator (*self.employee->select(e| e.age<25)->size()*) whereas the right child is the access to

the attribute *maxJuniors* with a child representing the access to the *self* variable. The rest of the constraint is represented in a similar way.

We use the method proposed in [1] to determine the set of PSEs that may violate an integrity constraint. This method draws those PSEs from the nodes of the OCL expression that defines the constraint by means of examining the elements and operations referred in the constraint as well as its syntactic structure.

As an example, we have that the application of this method over the tree representing the constraint *MaxJuniors* would result in the marked tree of Fig. 2. Each node is marked with the set of structural events that may violate the subexpression it represents. For instance, the access to the attribute *maxJuniors* is marked with *UpdateAttribute(maxJuniors,Department)* and *InsertET(Department)* since an update of the attribute *maxJuniors* (in particular a decrease of its value) or the creation of a new department (with more junior employees than permitted) may violate the constraint. Other PSEs for *MaxJuniors* are: *InsertRT(WorksIn)* and *UpdateAttribute(Age,Employee)*.

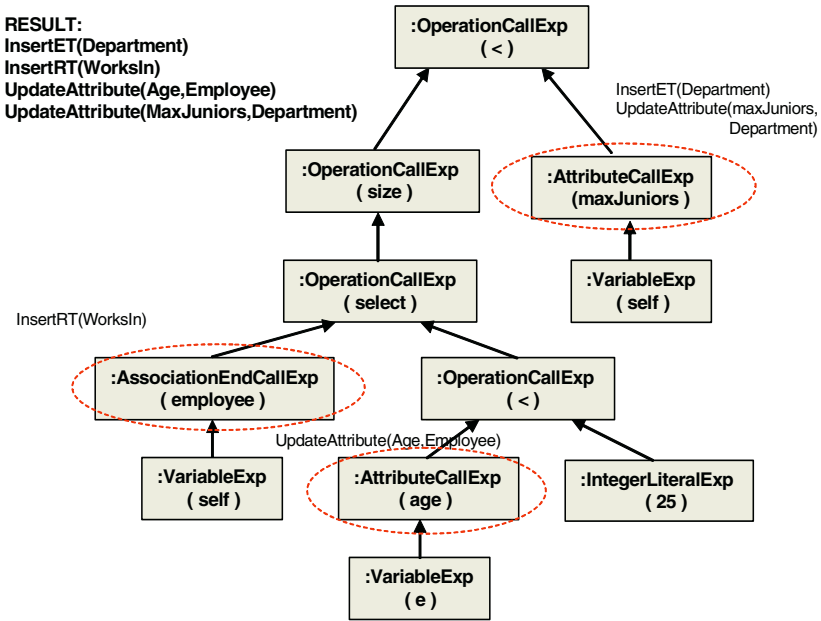


Fig. 2. Computing the set of PSEs for *MaxJuniors*

### 3 Constraint Classification

After executing a set of operations over the IB, we must verify all constraints having as PSEs some of the structural events included in them. A direct computation of the OCL expression defining a constraint would evaluate it over all instances of the context entity type (CET). However, this is not always necessary and many times we

can check a constraint by considering only the relevant instances of its CET (those affected by the set of structural events).

For instance, consider again the constraint *MaxJuniors*. After changing the age of an employee, instead of checking all departments, we only need to verify the departments where the modified employee works in.

In general, we may distinguish three different types of integrity constraints: *instance*, *partial instance* and *class* constraints. We classify a constraint as instance if we can always compute the exact subset of the instances of its CET we need to take into account to check it. A constraint is a class constraint if we always have to consider the whole population of the CET to check the constraint. Finally, in some cases we may need to consider the whole CET population or just a subset depending on the structural events issued during operation execution. In this case we say the constraint is partial instance.

*MaxJuniors* is a good example of instance constraint. We also have that *MaxFreelances* (*context Freelance inv: Freelance.allInstances()->size()<=100*) is a class constraint. The reason is that after inserting a new freelance we need to access all instances of the entity type *Freelance* to verify the number of freelances is still less than 101. Finally, *NumberEmployees* (*context Department inv: self.employee->size()<= Employee.allInstances()->size()/2*) is a partial instance constraint. Note that if we assign a new employee to a department, we only need to check the constraint over that particular department. However, if we remove an employee, we need to verify all departments, including those where the removed employee did not work.

A constraint can be classified into exactly one of those types just by examining the syntactic structure of the OCL expression defined in the body of the constraint. Intuitively, a constraint will be classified as *instance* if it is defined by means of a contextual instance (i.e. using, implicitly or explicitly, the *self* variable). A constraint will be a *class constraint* if it is defined using the *allInstances* operation. A *partial instance constraint* is a constraint that includes in its definition both the *self* variable and the *allInstances* operation.

To formally classify a constraint within the above categories, we need to introduce the concept of subexpression. In short, a subexpression is a sequence of nodes of the tree representing the OCL expression of the constraint. An OCL expression can consist of several subexpressions. In particular, each node representing an access to a variable begins a different subexpression. The reference to an entity type that precedes the *allInstances* operation is also considered a variable

Then, we can define a subexpression as the sequence of nodes that starts with this initial node and includes all its consecutive parent nodes that are traversed up to the last node of the subexpression. The last node is a node whose parent represents a call to an arithmetic operation, arithmetic or boolean comparison or a loop expression having the last node as its right child.

Fig. 3 shows the different subexpressions of the *MaxJuniors* and *Number Employees* constraints. An ellipse circles each subexpression.

We distinguish between two kinds of subexpressions: *instance* and *class* ones. A subexpression is considered an instance subexpression when it begins, directly or indirectly, with the *self* variable. A subexpression begins indirectly with the *self* variable when begins with a variable  $v$  where  $v \triangleleft self$  and  $v$  is defined within a loop



expression (*select*, *forall*...) included in an instance subexpression. Otherwise, the subexpression is considered a class subexpression. The same Fig. 3 classifies each subexpression for the example constraints.

Given a constraint  $c$  we define that  $c$  is an *instance constraint* when all the subexpressions of  $c$  including nodes with PSEs are instance subexpressions. We define that  $c$  is a *class constraint* when all the subexpressions of  $c$  including nodes with PSEs are class subexpressions. Finally, we define  $c$  as a *partial instance constraint* when it is neither an instance constraint nor a class constraint, and thus,  $c$  includes an instance subexpression and a class subexpression, at least. Our method improves the verification of instance and partial instance constraints but not the verification of class constraints (where we always need to examine all the instances).

Applying the previous definitions over the example constraints (Fig. 3) we obtain that the constraint *MaxJuniors* is an instance constraint and *NumberEmployees* is a partial instance constraint.

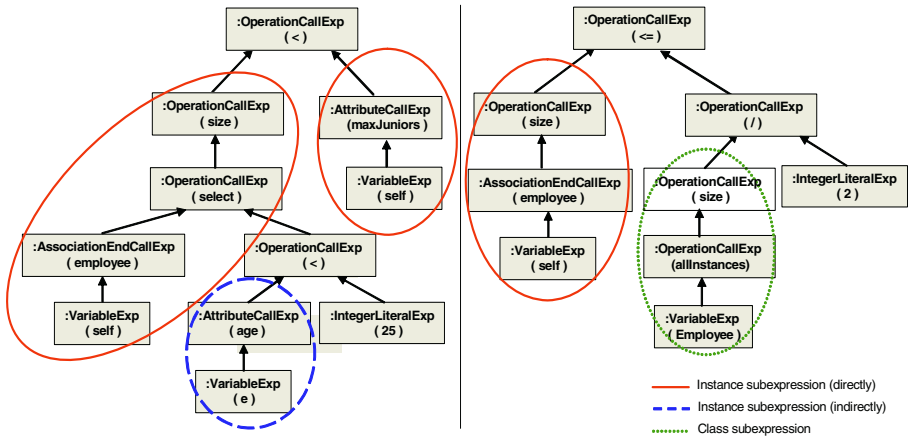


Fig. 3. Subexpressions for MaxJuniors (left) and NumberEmployees (right) constraints

## 4 Processing Instance Constraints

We explain now the transformation we propose for *instance constraints* to evaluate them only over the relevant instances of the IB. As we have just seen, *instance constraints* must only be evaluated over those instances of the context entity type that may have been affected due to the structural events issued during the transaction, since these are the only instances that can violate the constraint.

Given a constraint  $c$  defined over a context entity type CET, the basic idea of our transformation process is to create a new derived entity type meant to contain only those instances of the CET that need to be evaluated. More specifically, the population of the new type will be the set of instances of CET affected by the structural events. To compute its population we record that set of structural events in a special kind of entity types, the *structural event types*. Then, the context of the

constraint  $c$  is changed from CET to the new derived type, and thus, the constraint is evaluated only over the relevant instances.

In the following, section 4.1 explains the definition and treatment of the structural event types while section 4.2 explains the creation of the new derived entity type, the computation of its population and the redefinition of the constraint.

## 4.1 Definition of Structural Event Types

We need to define *structural event types* to record explicitly the structural events. More concretely, these types are devoted to record the information about the modifications produced by the structural events issued during the transaction.

In general, we need to define a *structural event type* for each possible structural event. Therefore we define the following types for each entity type of the CS (see section 2): iET (to record insertion events over the entity type  $ET$ ), dET (for deletion events over the entity type  $ET$ ), gET (a generalize event over  $ET$ ) and sET (an specialize event over  $ET$ ). Additionally, for each attribute of  $ET$ , we define a structural event type uETAttribute to record the changes in the attribute value. Moreover, for each relationship type  $RT$  we need to define: iRT (insertion of a new link in  $RT$ ) and dRT (a deletion of a link of  $RT$ ).

Nevertheless, since we simply use these types for dealing with instance constraints, we are only interested in defining the types corresponding to structural events that may be a PSE for that kind of constraints. Therefore, if a structural event cannot violate any of the ICs of the CS, we do not define its corresponding structural event type.

As an example, the list of structural event types we will define for the constraint *MaxJuniors*, according to its set of PSEs (see section 2), is the following: *iDepartment* (insertion of a new department), *iWorksIn* (insertion in the relationship *WorksIn*), *uDepartmentMaxJuniors* (update of the attribute *MaxJuniors*), and *uEmployeeAge* (update of the attribute *age*).

Note that we never need to define a structural event type for deletion events over entity types since this event is never included in the set of PSEs of an instance constraint. In fact, this kind of structural event can never appear in an instance subexpression. Since an instance subexpression begins (directly or indirectly) with the *self* variable, it is obvious that can not contain the event deleteET when  $ET=CET$  (we only evaluate the constraint over existing instances). Moreover, when  $ET \neq CET$ ,  $ET$  is accessed from a navigational expression starting with the *self* variable. In such cases it is the deletion of the link between the instance of  $ET$  and the previous instance in the navigation that can violate the constraint, not the deletion of the instance itself.

### 4.1.1 Structure of Structural Event Types

The next question we need to ponder is the internal structure (attributes and relationship types) of the structural event types. They are stereotyped with the stereotype <<structural event>> to differentiate them from the entity types of the CS.

We distinguish between structural event types recording structural events that modify entity types and those that modify relationship types.

The structural event types recording structural events that modify entity types are defined as types without attributes and with just one relationship type relating the

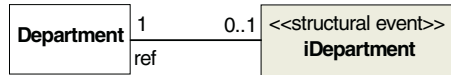
structural event type with the corresponding entity type. Through this reference we can access the entity modified by the structural event

The multiplicity of the relationship type between the structural event type and the entity type is 0..1:1. The reason is that an instance of the structural event type must necessarily refer to an instance of its entity type while an instance of the entity type may appear, at most once, in a structural event type. For the sake of simplicity, the role next to the entity type in all those relationship types is always named as *ref*.

Fig. 4. shows, as an example, the structural event type for the event insertET (Department). Note that the only information recorded for each instance of *iDepartment* is a reference to the corresponding new department instance in the *Department* type to access its information when needed.

We can opt for this kind of structure because there are no structural event types for deletion events (see section 4.1), and thus, we can always relate the instance of the structural event type with the corresponding entity in the entity type.

In the definition of these types we assume that the IB corresponding to the CS is updated at execution time with the modifications produced by the structural events. Thus, we can avoid redundancies by not including in the structural event type the information about the changes produced by the event over the affected entity (i.e. in the type *iDepartment* we do not include the information about the attribute values of the new department, we just use the reference towards the *Department* type to obtain this information).



**Fig. 4.** Structural event type for the event insertET over *Department*

In a similar way, the structural event types for structural events over relationship types do not contain attributes either. However, their instances do not refer to the corresponding link of the relationship type but to the set of participants of that link.

Therefore, a structural event type corresponding to a structural event over a relationship type *RT*, contains as many relationship types as the number of participants in *RT*. Each one of these relationship types relates the structural event type with one of the participants of *RT*. Note that the types dRT (deletion of a link of *RT*) are perfectly possible since their instances do not point to the deleted link (which does not already exist in the IB) but to their participants.

As an example, Fig. 5 shows the structural event type for the event insertRT(WorksIn). The type *iWorksIn* presents two relationship types, with *Department* and *Employee*, since these entity types are the participants of *WorksIn*.

When defining the multiplicity of the relationship types between the structural event type and the set of participants we distinguish between types for deletion events (dRT) and types for insertion (iRT) events.

For iRT types, the multiplicity of the relationship type is 1:\* since, in general, an entity of a participant entity type can participate in many links of the relationship type (for instance, if we assign a set of employees to the same department, several

instances of *iWorksIn* will refer to the same department entity) and every instance of the iRT type must be related to an existing entity of the participant entity type.

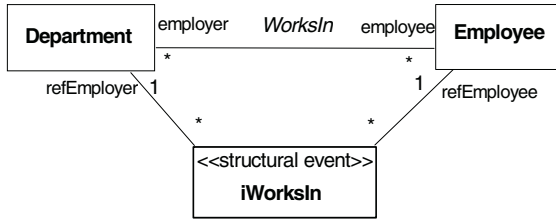


Fig. 5. Structural event type for the event insertRT over WorksIn

For dRT types, the multiplicity may become 0..1:\*, because, after deleting the link, and thus, creating a new instance in the dRT type, it may happen that other events delete also some of the participants of the link. This is not possible for iRT types since we cannot delete the participant without deleting before the link itself.

Note that we cannot remove the instance of dRT when deleting one of the participants since we may still need the information about the deleted link to compute the relevant instances for constraints including the deletion event as PSE. We can only delete it when all participants are deleted.

The constraints including as a PSE the event deleteRT over a relationship type *R*, either navigate *R* from *E<sub>1</sub>* to *E<sub>2</sub>* or from *E<sub>2</sub>* to *E<sub>1</sub>*, where *E<sub>1</sub>* and *E<sub>2</sub>* are the participant entity types of *R*. When, after deleting a link of *R*, the participant *E<sub>1</sub>* is also deleted, the information about the deleted link is irrelevant for constraints that navigate *R* from *E<sub>1</sub>* to *E<sub>2</sub>*. In such a case, it is the deletion of *E<sub>1</sub>* what must be taken into account. However, for constraints navigating *R* from *E<sub>2</sub>* to *E<sub>1</sub>*, the deleted link is used to navigate through the affected *E<sub>2</sub>* participant to obtain the relevant instances for the constraint.

For instance, consider a constraint stating that all departments must have at least three employees. The constraint can be violated by a deletion over *WorksIn*. If we delete the link between a department *d* and an employee *e*, a new instance of *dWorksIn* is created. Even if, afterwards, we also delete the employee *e*, the instance of *dWorksIn* allows us to know that the department *d* needs to be considered when evaluating the constraint.

#### 4.1.2 Instantiating the Structural Event Types

In general, each structural event type will contain as many instances as events of that kind have been executed over the entity or relationship type. For instance, the structural event type *iDepartment* will contain an instance for each new department inserted during the transaction, *uEmployeeAge* an instance for each employee that has changed its age during the transaction, etc.

However, to improve the efficiency of these types we adapt the concept of *net effect* [3] and define two additional rules for insertions and deletions over structural event types:

- Before inserting an instance in an `uETAttribute` type we must check that the same instance does not appear previously in the types `iET` or `uETAttribute`, as well. For instance, if we update three times the attribute *age* of the same employee during a single transaction, we only record this fact once.
- When deleting an entity or a relation, the corresponding instance is also deleted from the types `iET` (`iRT`), `gET`, `sET` and `uETAttribute` if existing. In addition, if the entity (relation) appears in `iET` (`iRT`) we do not need to record that it has been deleted. For instance, if we update the age of an employee and later on, during the same transaction, we delete the employee we do not need to worry about its age update. If the employee was inserted in the same transaction we neither record his/her deletion.

## 4.2 Constraint Redefinition

As we said before, to evaluate an instance constraint only over the relevant instances of its CET we create a new derived entity type meant to contain the exact set of instances of CET that need to be verified.

This new entity type, called *ETConstraint* (i.e. the name of the entity type plus the name of the constraint) is defined as a derived subtype of CET. Then, we replace the original constraint with a new constraint with the same body but having as a context entity type the new type *ETConstraint*. This is possible because, as a subtype, *ETConstraint* contains all attributes and relationship types of its supertype. As an example, Fig. 6 includes the redefinition of the constraint *MaxJuniors* over the *Department* entity type.

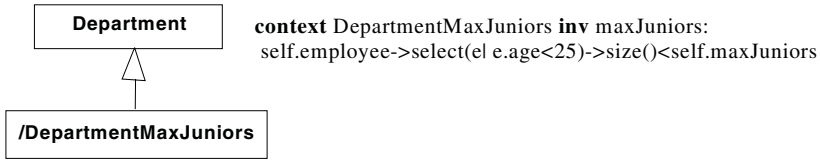
Note that with this replacement we obtain an efficient evaluation of the constraint, since the population of *ETConstraint* is exactly the set of instances we need to check. In general, the cardinality of the *ETConstraint* type is, by far, lesser than the total number of instances of CET and it can never be greater. Moreover, this approach also avoids redundant checking. The population of an entity type is a set and this ensures that we check each instance only once even if the transaction includes several structural events that affect the same instance.

The last problem we need to address is the computation of the population of the *ETConstraint* entity type, i.e. how to automatically define its derivation rule using the set of structural events recorded in the structural event types explained in the previous section. In short, the population of *ETConstraint* is the union of instances of CET affected by each structural event appearing in the structural event types. Obviously, if the structural event is not a PSE for the constraint no instances of CET are affected.

### 4.2.1 Computing the Instances of CET Affected by a Structural Event

Intuitively, given an instance *i* of a structural event type *ev* over the entity type *ET*, we obtain the set of instances of CET affected by *i* by doing an inverse navigation from *I* to the instances of CET related with *i*. Roughly speaking, the inverse navigation involves four different steps:

- To select the subexpression of the constraint where the event *ev* is included in.
- To reverse the part of the subexpression affected by the event. We reverse the part of the subexpression that goes from the beginning of the subexpression up to the element affected by the event *ev*.



**Fig. 6.** Redefinition of the *MaxJuniors* constraint

- To remove from the previous result all the elements except for the navigations over relationship types.
- For each navigation appearing in the reversed subexpression, to navigate through the same relationship type but in the opposite direction by means of using the opposite role.

As an example, consider the event `UpdateAttribute(Age,Employee)` over the constraint *MaxJuniors* (*context department inv: self.employee->select(e| e.age<25)->size()<self.maxJuniors*). This event affects the *select* operation of the constraint so we need to take into account the subexpression *self.employee->select(e|e.age<25)*. The subexpression only contains the navigation through the relationship type *WorksIn* using the *employee* role. Therefore, to obtain the affected departments after the age update, we just navigate from the updated employee to the departments related with him through the same *WorksIn* relationship type but using the opposite role (the *employer* role).

Formally, assume a constraint defined over a CET with a PSE attached to the node  $n_i$  of the tree representing the constraint and where  $n_i$  is included, at least, in an instance subexpression *sub* where  $sub = [n_0, n_1, n_2, \dots, n_{i-1}, n_i, n_{i+1}, \dots, n_n]$ .  $n_0$  is the initial node of the subexpression and  $n_n$  the last one. We obtain the set of instances of CET affected by an execution of the PSE by, first, reversing the sequence of nodes  $[n_0, n_1, n_2, \dots, n_{i-1}, n_i]$  to obtain the sequence  $[n_i, n_{i-1}, \dots, n_2, n_1, n_0]$ . Note that the reversed sequence starts with the node responsible for the PSE.

Then, we delete from the sequence all nodes that do not represent a navigation through a relationship type (i.e. all nodes not representing an access to an association end). Finally, for each remaining node, we replace the node with another node representing an access to the opposite association end.

When the node  $n_i$  appears in an indirectly instance subexpression *sub* we need to concatenate the nodes of *sub* with those of its *parent* subexpression and repeat the process until we reach the node representing the initial *self* variable. The *parent* subexpression is the subexpression containing the iterator where *sub* is included. More concretely, if the parent subexpression is of the form  $parent = [p_0, p_1, p_2, \dots, p_{i-1}, it, p_{i+1}, \dots, p_n]$  where *it* is the iterator where *sub* is included in, the result of the concatenation is  $result = [p_0, p_1, p_2, \dots, p_{i-1}, n_0, n_1, \dots, n_{i-1}, n_i]$ , and after reversing the order of nodes  $[n_i, n_{i-1}, \dots, n_2, n_1, n_0, p_{i-1}, \dots, p_2, p_1, p_0]$ .

If the same PSE appears in different instance subexpressions or the node  $n_i$  is included in several ones we repeat the process for each of them.

As an example, we apply the formalization to obtain the set of departments we need to check in the *MaxJuniors* constraint after the event `UpdateAttribute(Age,Employee)`. The subexpression *sub* where the event is included is  $sub = [e, age]$

(see Fig. 3.). Since this is an indirectly instance subexpression we must concatenate it with its parent subexpression ( $parent=[self, employee, select, size]$ ). The resulting subexpression is  $[self, employee, e, age]$ , and once reversed  $[age, e, employee, self]$ . We remove all the irrelevant nodes to obtain the sequence  $[employee]$  and, once replacing the node by the opposite role, we obtain the final result  $[employer]$ , where  $employer$  is the opposite association end of  $employee$ , the only node representing an access to an association end.

Therefore, to obtain the affected departments we need to apply the obtained subexpression ( $[employer]$ ) over each updated employee (i.e. each instance of  $uEmployeeAge$ ). For instance, if the  $uEmployeeAge$  type contains an instance  $e1$ , we access the updated employee using  $e1.ref$  (see section 4.1) and then we obtain the affected departments using the expression  $e1.ref.employer$ .

As a more complex example, consider a constraint stating that an employee cannot be older than the bosses of the departments where he/she works. This constraint could be expressed as:  $context Employee inv: self.employer->forall(d| d.boss.age>self.age)$ . When computing the set of PSEs for the constraint we see that the event  $UpdateAttribute(Age,Employee)$  is included in both constraint subexpressions. Thus, to obtain the set of employees we need to check after an age update, we have to apply the previous process over both subexpressions and join the two sets of affected employees.

The first subexpression is  $[self, employer, d, boss, age]$ , and once reversed  $[age,boss,d,employer,self]$ . After removing the irrelevant nodes:  $[boss,employer]$  and the final result, once replacing the nodes with the opposite association ends, is  $[managed,employee]$ . Therefore, to obtain the affected employees we need to navigate from the updated employee to the department he/she manages (if any), and then, from the department to the employees of that department.

The second subexpression is  $[self,age]$ . It does not include any navigation, and thus, the final result will be an empty sequence of nodes. This means that given an updated employee, we only need to check that particular employee.

As a final result we obtain that after an age update we need to check the updated employee plus all the employees working in the department he/she manages, if any.

#### 4.2.2 Derivation Rule Definition

The derivation rule for the  $ETConstraint$  entity type must ensure that the set of instances of the type be exactly the set of instances we need to check. It must include, for each instance of the structural event types corresponding to the PSEs for the constraint, the computation of the affected CET instances, as explained above. Using the work of [8] we define the population of a derived entity type by means of redefining its predefined  $allInstances$  operation (i.e. the population of the derived type will be the set of instances returned by the operation).

As an example, consider the previous  $MaxJuniors$  constraint. In this case, the derivation rule for the derived subtype  $DepartmentMaxJuniors$  must select, according to the PSEs for the constraint, all new inserted departments (departments recorded in the  $iDepartment$  structural event type), the departments that have updated its  $maxJuniors$  attribute (departments appearing in the  $uDepartmentMaxJuniors$  type) and the departments with new assigned employees (departments participating in a new relationship of the  $WorksIn$  relationship type, recorded in the  $iWorksIn$  type), and



also, for each employee that has changed his/her age, all the departments where the employee was working in.

This last set of departments is obtained by applying the role *employer* over each updated employee (each instance of the *uEmployeeAge* type) as computed in the previous section.

Therefore, the derivation rule for *DepartmentMaxJuniors* is the following:

**context** *DepartmentMaxJuniors::allInstances()* : *Set(Department)*

**body:** *iDepartment.allInstances().ref->union(*  
*uDepartmentMaxJuniors.allInstances().ref->union(*  
*iWorksIn.allInstances().refEmployer->union(*  
*uEmployeeAge.allInstances().ref.employer)))->asSet()*

Note that, we use the special relationship types between the structural event types and its corresponding entity types to access the modified instances (see section 4.1). For instance, *iDepartment.allInstances().ref*, returns the new departments by accessing the referenced departments from the *iDepartment* type.

In [2] we show the results of the application of our method over the rest of instance constraints of our example.

## 5 Processing Partial Instance Constraints

A constraint is classified as a partial instance constraint if it contains at least an instance subexpression and a class subexpression, both including nodes marked with PSEs for the constraint. These constraints can be checked efficiently when the transaction does not include any of the PSEs included in class subexpressions. Otherwise, we must check the constraint over all instances of the CET. For instance, the constraint *NumberEmployees* (*context Department inv: self.employee->size() < Employee.allInstances()->size()/2*) can be checked efficiently after assigning an employee to a department but not after the deletion of an employee.

To process this kind of constraints we split their set of PSEs into two different groups: the set of *instance* PSEs and the set of *class* PSEs, depending on the kind of subexpression where they are included. If a PSE is included in both kinds of subexpressions is considered a class PSE. With the set of instance PSEs we apply exactly the same process explained in section 4 with just a slight difference concerning the derivation rule of the *ETConstraint* entity type, as we explain below.

For the *class* set we also create the structural event types. In fact, we are not interested in knowing the exact instances affected by the class PSEs because we will need to check all instances. We only need to know whether any of the those events has been executed, and thus, we could think about creating a new set of singleton entity types enough to record the presence or absence of each event. However, since probably most structural event types will be already defined to deal with other constraints, we think it is worthwhile to reuse the same set of structural event types.

The only difference relies on the dET kind of entity types, which were not needed before. If there is a deleteET event among the set of *class* PSEs, we need to create the corresponding dET type. As we have explained before, the instances of this entity type cannot reference the deleted instances since they no longer exist, but this



does not suppose a problem since we are not interested in knowing those instances. We just create an empty instance in the dET type.

Afterwards, in a similar way as before, we create a new derived subtype, called *ETConstraint'*, under the context entity type, and change the context of the original constraint to *ETConstraint'*. Its population will be the same population of the context entity type if the transaction has executed any class PSE. Otherwise, its population will be empty. Therefore, the derivation rule for *ETConstraint'* is:

$allInstances() = \mathbf{if} ( ev_1.allInstances()->size() + ev_2.allInstances()->size() + \dots ev_n.allInstances()->size() > 0) \mathbf{then} CET.allInstances()$   
 where  $ev_1..ev_n$  represent the structural event types corresponding to the class PSEs.

Moreover, we change the derivation rule *dr* for the *ETConstraint* type created for the instance PSEs. The new derivation rule will be:  $allInstances() = \mathbf{if} (ETConstraint'.allInstances()->size()=0) \mathbf{then} dr$ . This way we ensure that when the transaction includes a class PSE we check all instances of the original context entity type (*ETConstraint'* will contain the same instances as CET) and avoid redundant checkings (*ETConstraint* will be empty). Otherwise, we check the constraint efficiently (*ETConstraint* will contain the affected instances of CET whereas *ETConstraint'* will be empty).

In [2] we process the partial instance constraint *NumberEmployees*.

## 6 Conclusions and Further Work

We have proposed a new method to improve efficiency of integrity constraint checking in CSs specified in UML with constraints written in OCL. As far as we know, ours is the first method to deal with this issue at the specification level.

The basic idea of our method is to record the set of structural events applied over the IB in order to compute the set of relevant instances for each constraint, and then, evaluate the constraint only over those instances, avoiding irrelevant verifications.

We believe this efficiency gain justifies the overhead of computing the set of relevant instances, since, in general, the number of relevant instances will be much lesser than the total number of instances. Moreover, the number of entity types added to the CS is limited since structural event types do not depend on the number of constraints in the CS and for each constraint at most two derived types are defined.

Our method uses only the standard elements of any conceptual schema (entity types, relationship types, derived elements and integrity constraints) to transform the original constraints into efficient ones. Therefore, the results of our method can benefit any implementation of the CS, regardless the technology used. In fact, any code-generation strategy able to generate code from a CS could be enhanced with our method to automatically generate efficient constraints, with only minor adaptations.

The expressive power of the OCL language permits to write the same semantic constraint in a variety of syntactic forms. Since our method relies on the syntactic definition of the OCL expressions, choosing the simplest representation of a constraint can entail a more efficient verification of the constraint. As further work, we plan to study how to automatically transform a constraint definition into its simplest representation to guarantee the best results when applying our method.

## Acknowledgements

We would like to thank people of the GMC group for their many useful comments to previous drafts of this paper. This work has been partially supported by the Ministerio de Ciencia y Tecnología and FEDER under project TIC2002-00744.

## References

1. Cabot, J., Teniente, E.: Determining the Structural Events that May Violate an Integrity Constraint. In: Proc. of the 7<sup>th</sup> UML Conference (UML'04) , LNCS 3273, pp. 320-334
2. Cabot, J., Teniente, E.: Computing the Relevant Instances that May Violate an OCL constraint. LSI Research Report, LSI-05-5-R, UPC, 2005
3. Ceri, S., Widom, J.: Deriving Production Rules for Constraint Maintenance. In: Proc. 16th VLDB Conference (VLDB'90), Morgan Kauggmann, pp. 566-577
4. Gogolla, M., Richters, M.: Expressing UML Class Diagrams Properties with OCL. In: A. Clark and J. Warmer, (eds.): Object Modeling with the OCL. Springer, 2002, pp. 85-114
5. Gupta, A., Mumick, I. S.: Maintenance of materialized views: problems, techniques, and applications. In: Materialized Views Techniques, Implementations, and Applications. The MIT Press, 1999, 145-157
6. ISO/TC97/SC5/WG3: Concepts and Terminology for the Conceptual Schema and Information Base. ISO, 1982
7. Olivé, A.: Time and Change in Conceptual Modeling of Information Systems. In: S. Brinkkemper, E. Lindencrona, and A. Solvberg, (eds.): Information Systems Engineering. State of the Art and Research Themes. Springer, 2000, pp. 289-304
8. Olivé, A.: Derivation Rules in Object-Oriented Conceptual Modeling Languages. In: Proc. 15<sup>th</sup> Int. Conf. on Advanced Information Systems Engineering (CAISE'03), LNCS, 2681, pp. 404-420
9. OMG: UML 2.0 OCL Specification. OMG Adopted Specification (ptc/03-10-14), 2003
10. OMG: UML 2.0 Superstructure Specification. OMG Adopted Specification (ptc/03-08-02), 2003
11. Wieringa, R.: A survey of structured and object-oriented software specification methods and techniques. ACM Computing Surveys 30, 1998, pp. 459-527

# Event-Based Modeling of Evolution for Semantic-Driven Systems

Peter Plessers<sup>\*</sup>, Olga De Troyer, and Sven Casteleyn

Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussel, Belgium

{Peter.Plessers, Olga.DeTroyer, Sven.Casteleyn}@vub.ac.be

**Abstract.** Ontologies play a key role in the realization of the Semantic Web. An ontology is used as an explicit specification of a shared conceptualization of a given domain. When such a domain evolves, the describing ontology needs to evolve too. In this paper, we present an approach that allows tracing evolution on the instance level. We use event types as an abstraction mechanism to define the semantics of changes. Furthermore, we introduce a new event-based approach to keep depending artifacts consistent with a changing instance base.

## 1 Introduction

The Semantic Web is an extension of the current Web; information is given a well-defined meaning better enabling computers and people to work in cooperation [1]. Ontologies play a major role in the Semantic Web where the meaning of web content is formalized by means of ontologies. An ontology is defined as an explicit specification of a shared conceptualization [2]. We also see that other research domains are adopting technologies developed within the Semantic Web domain. E.g. ontologies are used in content and document management, information integration and knowledge management systems to provide extensive reasoning capabilities, intelligent query possibilities, integration and cooperation between systems, etc. We will use the term “semantic-driven” systems to refer to such systems.

Systems and their environments are not static but evolve. Domains evolve and changes in user requirements occur often. To keep semantic-driven systems up to date, changes in the environment should be reflected in the underlying ontology. Furthermore, also flaws and errors in the design of the ontologies may call for a revision of the ontology. Manual handling of the evolution process of ontologies as well as managing the impact of evolution on depending ontologies, instance bases, applications and annotations, is not feasible because it would be too laborious, time intensive and complex. Therefore, an automatic mechanism should be provided.

Ontology evolution takes place on two levels: on the instance level (e.g. a ‘prime minister’ who resigns after an election defeat and becomes a ‘senator’) and on the

---

<sup>\*</sup> This research is partially performed in the context of the e-VRT Advanced Media project (funded by the Flemish government) which consists of a joint collaboration between VRT, VUB, UG, and IMEC.

concept level (e.g. a new rule that forbids people to be candidate for more than one parliament). Current approaches for supporting ontology evolution [3], [4] propose a single approach dealing with evolution on the instance as well as on the concept level. In this paper, we concentrate on evolution on the instance level (evolution of the concept level is outside the scope of this paper) and the impact of this evolution on depending artifacts. Our approach delivers a number of advantages not found in existing approaches.

The remainder of the article is organized as follows. In section 2 we give a short overview of related work. A general overview of our approach is given in section 3, more details are given in section 4. The approach is further elaborated in section 5 (time aspect) and section 6 (events). Section 7 explains the handling of the impact of evolution for depending artifacts. Section 8 ends the paper with conclusions.

## 2 Related Work

The work presented in this paper is related to the fields of temporal databases and ontology evolution.

Conventional databases capture the most recent data. As new values become available, the existing data values are removed from the database. Such databases only capture a snapshot of reality and are therefore insufficient for those in which past and/or future data are required [5]. Temporal databases [6] typically allow differentiating between temporal and non-temporal attributes. The temporal database maintains a state history of temporal attributes using time stamps to specify the time during which a temporal attribute's value is valid.

Different conceptual models that support the modeling of temporal databases exist. They can be divided into three categories: extensions to relational data models (e.g. TER [7], TERM [8] and ERT [9]), object-oriented approaches (e.g. TOODM [10]) and event-based models (e.g. TEERM [11]). The approach described in this paper resembles most the philosophy taken by event-based models. These models don't record past states of a system, but rather events that change the state. Note that in these models events are just 'labels' i.e. they don't define the meaning of the event.

Ontology evolution approaches [3], [4] propose methods to cope with ontology changes and techniques to maintain consistency of depending artifacts. Both approaches present a meta-ontology to represent changes between ontology versions. A log of changes is constructed in terms of this meta-ontology. Consistency is maintained by propagating changes (listed in the log of changes) to depending artifacts.

The difference with our approach is that we formally define the semantics of changes (by means of event types). This allows us to reason about changes on a higher level of abstraction. Furthermore, event types are used to maintain consistency between an instance base and depending artifacts.

## 3 Approach: Overview

To keep track of changes to the instance base, we use the following approach. Whenever a change is made to the instance base, the log of changes is updated. The log is

defined in terms of the *evolution ontology*. Third-party users can use this log structure to check if a (for their artifact) relevant change occurred. To automate this, third-party users can specify a set of *event types* in which they are interested. If after the update one or more instances satisfy the definition of one of their event types, an instance of this event type will be created. The event type is an abstraction mechanism that allows us to reason about changes on a higher level of abstraction than possible with the log of changes. The event types and the events itself are captured in a log of events defined in terms of an *event ontology*.

After a change, the instance base and depending artifacts may be in an inconsistent state. Instead of forcing third-party users to upgrade their dependent artifacts to maintain consistency (as is done in other approaches), we present a technique based on event types that allows to validate if a dependency is still valid in the current state of the instance base and if not to refer to a previous state of the instance base. This approach allows third-party users to update at their own pace (if ever) without causing inconsistencies in the meantime.

Before proceeding with a more detailed description of our approach, we first introduce an example situation that will be used throughout the paper. We have constructed a small domain ontology describing the political domain of a federal state: its governments, parliaments, ministers, parliamentarians, etc. Also assume an instance base based on this ontology to populate the web portal of the government. Furthermore, there exists a third-party website listing the current and past ministers of the governments. The content of the website is annotated using the political instance base. Note that the owner of the instance base and the owner of the website are not necessarily the same. Moreover, the instance base does not support evolution as it is of no direct benefit for the government. Also note that the government is not necessarily aware of third-party users making use of the instance base. In addition, a third-party user may only be interested in tracing evolution for a very specific part of the changes that occur. E.g. the owner of the website is only interested in tracing the evolution of ministers; i.e. he is not interested in parliamentarians.

## 4 Approach: Details

### 4.1 Assumptions

Our approach is based on the following assumptions:

1. The underlying domain ontology of the instance base is build up using classes, object properties (relations between classes) and datatype properties (relations between a class and a data type e.g. strings, integers, ...). All these are called *concepts*.
2. Concepts are identified by a unique identifier that uniquely identifies a given concept during its 'lifetime'.
3. There exists three operations that users can apply to update an instance base:
  - *Create*: a new instance is added;
  - *Retire*: an instance is removed;

- *Modify*: an instance is modified. E.g. an instance of a concept A evolves into an instance of a concept B; the value of a datatype property is changed; etc.

Note that we explicitly need the ‘Modify’ operator as we can’t treat it as the combination of a ‘Retire’ and ‘Create’ operator. When we modify an instance by removing the instance first and afterwards adding a new instance, we cannot assure that it is the ‘same’ instance as identifiers can be reused.

As these assumptions are based on common features most systems will satisfy them.

## 4.2 Evolution Ontology

As explained in the overview, a log of changes made to an instance base is maintained in terms of the *evolution ontology*. For every instance of a class in the instance base, a new unique instance is created in the log of changes. This instance has a reference to the original instance in the instance base (at least as long as this instance exists in the instance base). In addition, the log keeps track of all the changes that are made to that instance. This is done by means of the *Change* concept.

A Change is defined as a concept with the following properties (see fig 1): a reference to the instance to which it refers (*instanceOf*); the operation used to make the change (*hasOperation*); and a time stamp identifying the date and time of the change (*hasTransactionTime*). We have defined three types of Changes: a change to instances of a class (*ClassChange*), to instances of an object property (*ObjectPropertyChange*), and to instances of a datatype property (*DataTypePropertyChange*). For an *ObjectPropertyChange* and a *DataTypePropertyChange* we also keep track of respectively the target instance (*hasTargetInstance*) and the value of the changed property (*hasValue*).

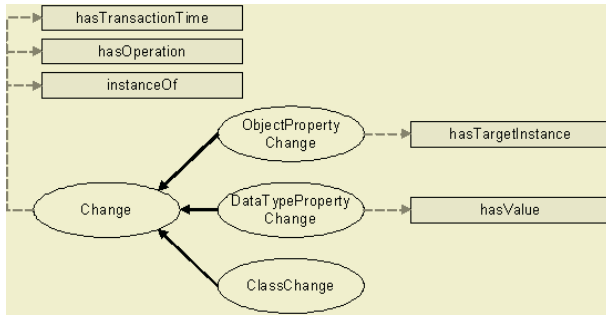


Fig. 1. Overview of Change classes

The following OWL code gives an example of a log of changes. It shows the changes applied to an instance 'john\_smith'. The first change denotes that 'john\_smith' is created as an instance of the concept 'Person' (see 1). Next, an instance of the datatype property 'hasName' was assigned with value 'John Smith' (see 2). Finally, he becomes politically active: 'john\_smith' changes to an instance of 'Politician' (see 3) and joins a political party (see 4).

```

<EvolutionClass rdf:ID="60f28870">
  <refersTo rdf:resource=".../politics#john_smith"/>
  <changeOccurred>
    <ClassChange rdf:ID="7ece6c60"> (1)
      <hasTransactionTime>07/05/03</hasTransactionTime>
      <instanceOf rdf:resource="#Person"/>
      <hasOperation rdf:resource="#Create"/>
    </ClassChange>
  </changeOccurred>
  <changeOccurred>
    <DataTypePropertyChange rdf:ID="7ece6c61"> (2)
      <hasValue>John Smith</hasValue>
      <hasOperation rdf:resource="#Create"/>
      <hasTransactionTime>08/05/03</hasTransactionTime>
      <instanceOf rdf:resource="#hasName"/>
    </DataTypePropertyChange>
  </changeOccurred>
  <changeOccurred>
    <ClassChange rdf:ID="7ece6c62"> (3)
      <hasTransactionTime>18/10/04</hasTransactionTime>
      <instanceOf rdf:resource="#Politician"/>
      <hasOperation rdf:resource="#Modify"/>
    </ClassChange>
  </changeOccurred>
  <changeOccurred>
    <ObjectPropertyChange rdf:ID="7ece6c63"> (4)
      <hasTarget rdf:resource="#political_party_x"/>
      <hasTransactionTime>19/10/04</hasTransactionTime>
      <hasOperation rdf:resource="#Create"/>
      <instanceOf rdf:resource="#memberOf"/>
    </ObjectPropertyChange>
  </changeOccurred>
</EvolutionClass>

```

As the use of operations trigger changes, this log can be generated automatically. Every operation to the instance base leads to a change in the evolution ontology indicating the difference between the current and previous state of an instance. Note that the reference to the original instance in the instance base is removed as soon as the instance retires from the instance base.

### 4.3 Event Ontology

The log of changes can be used by third-party users to get an overview of the changes that have occurred to the instance base they are using. This log of changes forms the basic mechanism to keep depending systems consistent with the changed instance base (see section 6). However, the approach also uses an *event ontology* because the evolution ontology does not allow to deal with the following situations:

- A third-party user may only be interested in particular changes. Take for instance our example instance base and annotated web page that lists the names of all current and past ministers. For such a page, we are interested in the event where a per-

son becomes a minister or retires as minister, but we don't care when a minister changes office, or is promoted to prime minister.

- Reasoning in terms of evolution is not convenient. Changes are defined at the lowest level and few semantics are captured because no reason or meaning of a change is given. E.g. what is the reason for deleting an instance of a concept 'Minister'? Was the minister fired? Did the government fall? Or was it just the end of his term?

To associate meaning to changes and to allow indicating relevant changes, a third-party user can define a set of event types. Event types are defined in terms of changes. In this way, events in the real world can be associated with changes in the log of changes. As an example, a third-party user may define an event type 'retireMinister' as the change of an instance from being an instance of the concept Minister to a retired instance.

Letting third-party users define their own set of relevant event types, allows tracing changes from different viewpoints. Although there exists a shared agreement concerning the domain ontology and its instance base, this doesn't mean that there is also a shared agreement about events. Therefore, different users may define different event ontologies. Moreover, an event type is an abstraction mechanism that allows to reason about changes on a higher level than possible with changes in the evolution ontology .

More details and a formal representation for event types will be given in section 6.

## 5 Time Aspect

An important aspect when tracing evolution is the notion of time. A linear time line  $T$  is therefore used. Changes (in the log of changes) as well as events (in the log of events) are linked to this time line by means of timestamps. These timestamps represent *transaction times*. Transaction time indicates when an instance was created, modified or retired from the instance base. For each change to an instance  $i$  we use the time line  $T$  to represent transaction times. This means that we define an explicit order on the changes for a particular instance  $i$ . This can be seen as an individual, relative time line. We refer to this time line as  $T_i$  where  $i$  is a given instance from the evolution ontology. A variable  $ct_i$  refers to the current time of an instance  $i$ , i.e. the moment in time the last change took place for this instance. If  $c$  ( $\in \mathbb{N}$ ) specifies the total amount of changes that occurred for an instance  $i$ , then we can use  $ct_i - a$  (where  $a \in \mathbb{N}$ ,  $a \leq c$ ) to refer to the moment in time the  $(c - a)^{\text{th}}$  change occurred for that instance.

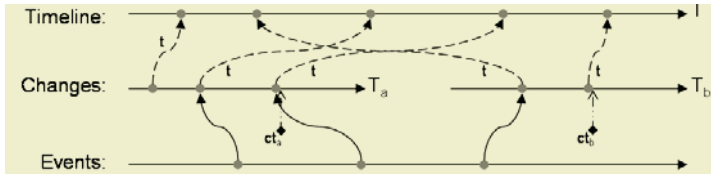
Events contain a reference to the change that triggered them. An event is indirectly linked to the time line  $T$  through the referred changes. Figure 2 gives an overview.

$T_i$  allows us to retrieve properties of instances relative to this time line. This means that we are able to request the value of a property for an instance at a certain moment in the past. The following notation is used to retrieve past states of instances:

```
<property_name>(<inst>, <value> | <var>, <timestamp>)
```

where  $\langle \text{inst} \rangle$  is an instance from the evolution ontology,  $\langle \text{value} \rangle$  is the value of a property while  $\langle \text{var} \rangle$  is a substitution, and  $\langle \text{timestamp} \rangle \in T_i$ . The  $\langle \text{timestamp} \rangle$  indicates the moment in time at which we request the property.





**Fig. 2.** Time aspect overview. 't' indicates transaction time

We give two examples. The first example checks if an instance  $i$  was an instance of the concept 'Minister' during the previous state of  $i$ . The second example retrieves the previous telephone number of the instance  $i$ ; the result is stored in a variable 'x'.

Example 1: `instOf(i, 'Minister', cti-1)`

Example 2: `hasTelephone(i, x, cti-1)`

A first step to resolve this query is to transform the abstract timestamp ( $\langle \text{timestamp} \rangle \in T_i$ ) into an absolute timestamp  $t \in T$ . Next, the past state of the instance base can be reconstructed by applying all stored changes that have occurred before the absolute time stamp. Finally, the query is resolved against the constructed state of the instance base.

## 6 Events

In this section we give more details about our event types. In section 6.1, we introduce basic event types. Basic event types are used to define the semantics of changes applied to one instance. We have defined a hierarchy of basic event types reflecting the meaning of basic changes. Users can subtype these basic event types to define their own set. As basic event types only define the meaning of changes to exactly one instance, we also have defined complex event types (see section 6.2) for changes involving more than one instance.

### 6.1 Basic Event Types

Figure 3 gives an overview of our basic event types. The root concept is the abstract class 'Event' and has three subclasses 'Creation', 'Modification' and 'Retirement'. These subclasses define the semantics of the changes resulting from the operations defined in section 4.1. The 'Modification' class is further refined into: 'Expansion', 'Contraction', 'Continuation', 'Extension' and 'Alteration'. The definitions of these event types are given in Figure 4.

To define the event types, we use the following definitions:

- The set  $I$  is the set of all instances of the evolution ontology (i.e. both class and property instances).

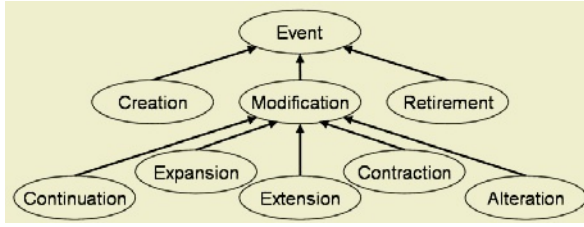


Fig. 3. Basic event type hierarchy

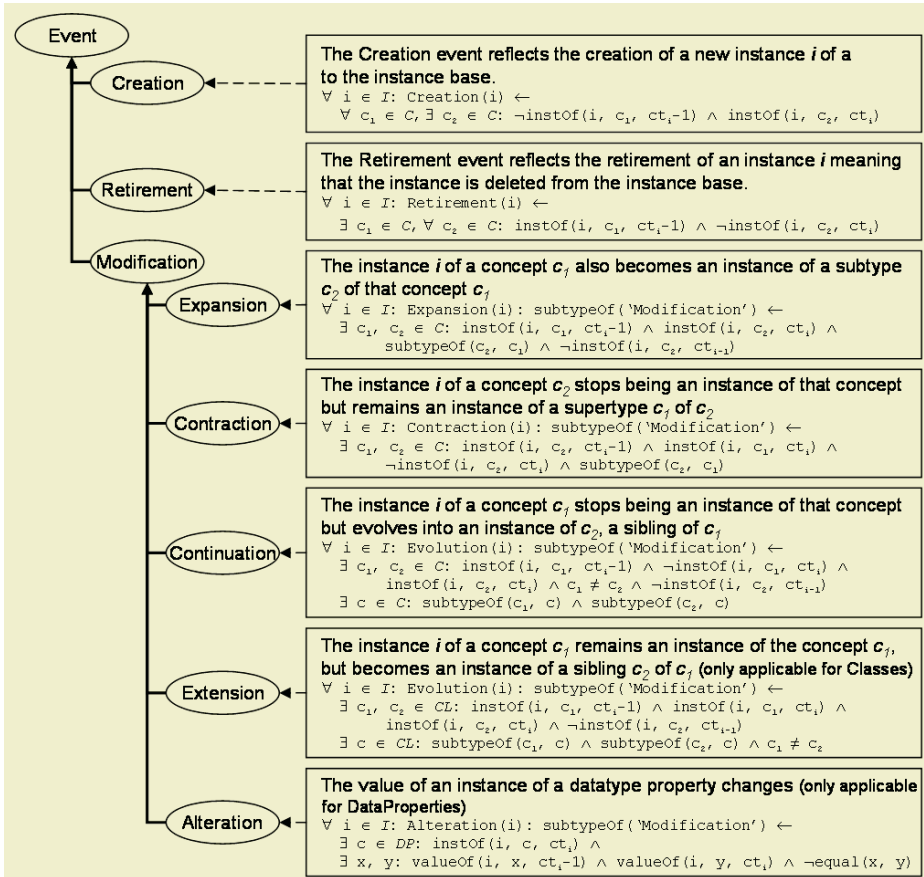


Fig. 4. Basic event type definitions

- The sets  $DP$ ,  $OP$  and  $CL$  are the sets of respectively all datatype properties, object properties and classes defined in the domain ontology.
- The set  $C = DP \cup OP \cup CL$ .

As an example event type, we define 'MinisterLeavesGovernment' that describes the change when a minister leaves a government. It is defined as a 'Retirement' event of an instance of the 'memberOf' property. The remainder of the definition checks if the source and target of the property were instances of respectively 'Minister' and 'Government'.

```

∀ i ∈ I: MinisterLeavesGovernment(i):
  subtypeOf('Retirement') ←
  instOf(i, 'memberOf', cti-1) ∧
  ∃ x, y ∈ I: source(i, x, cti-1) ∧ target(i, y, cti-1) ∧
    instOf(x, 'Minister', ctx) ∧
    instOf(y, 'Government', cty)

```

## 6.2 Composite Event Types

Basic event types define the semantics of a change of exactly one instance. This fine-grained type of events is not always sufficient. Often evolution on a higher level, taking into account changes to more than one instance, is necessary. Therefore, we also provide *composite event types*: event types that define the semantics of changes of more than one instance. We illustrate this with an example. The example defines the change where two ministers from different governments change places (e.g. a minister from a regional government switches to the federal government and a minister from the federal government goes to the regional government). First we define the basic event type 'MinisterChangesGovernment'. The event type defines the change where a minister leaves one government to join another one.

```

∀ i ∈ I: MinisterChangesGovernment(i):
  subtypeOf('Modification') ←
  ' i remains an instance of 'memberOf'
  instOf(i, 'memberOf', cti-1) ∧
  instOf(i, 'memberOf', cti) ∧

  ∃ x, y, z ∈ I:
    ' get the previous and current target and current
    ' source of i
    target(i, x, cti-1) ∧
    target(i, y, cti) ∧
    source(i, z, cti) ∧

    ' target and source are respectively instances of
    ' Government and Minister
    instOf(x, 'Government', ctx-1) ∧
    instOf(y, 'Government', cty) ∧
    instOf(z, 'Minister', ctz) ∧

    ' but the previous and current government are not
    ' the same instance
    ¬equal(x, y)

```

Second, we define a composite event type 'ExchangeOfMinisters' that makes use of the previously defined event type.

```

∀ i1, i2 ∈ I: ExchangeOfMinisters(i, j):
  subtypeOf('Event') ←

  ∃ t1 ∈ Ti, ∃ t2 ∈ Tj:
    ' the 'MinisterChangesGovernment' event occurred
    ' for both i and j
    occurredEvent1(i, 'MinisterChangesGovernment', t1) ∧
    occurredEvent(j, 'MinisterChangesGovernment', t2) ∧

    ' get governments
  ∃ g1, g2, old_g1, old_g2 ∈ I:
    ' previous government of i
    target(i, prev_g1, cti-1) ∧
    ' current government of i
    target(i, cur_g1, cti) ∧
    ' previous government of j
    target(j, prev_g2, ctj-1) ∧
    ' current government of j
    target(j, cur_g2, ctj) ∧

    'check governments
    equal(prev_g1, cur_g2) ∧
    equal(prev_g2, cur_g1)

```

The event first checks if the instances  $i$  and  $j$  both changed government in the past i.e. the 'MinisterChangesGovernment' event type should have occurred before. Next, we lookup the governments they both left and joined. The last two statements check if the two ministers swapped government. Note that we didn't put any time restriction on the occurrence of the 'MinisterChangesGovernment' event. If for instance, a minister leaves government  $a$  and joins government  $b$  and three years later another minister leaves government  $b$  and joins government  $a$ , these changes will match the definition of the 'ExchangeOfMinisters' event type. However, in this situation, we can hardly speak of an exchange. We could solve this issue by adding a time constraint to the event type definition stating that the exchange must occur within a time frame of for instance 2 months.

## 7 Consistency Between Instances and Depending Artifacts

A change to an instance remains mostly not restricted to that single instance, but may have an impact on related instances and depending artifacts. It could bring the complete system (i.e. instance base and depending artifacts) into an inconsistent state. It is a major requirement for any evolution approach to assure that the complete system evolves from one consistent state to another.

---

<sup>1</sup> OccurredEvent( $i, e, t$ ) checks if an event type  $e$  has occurred for an instance  $i$  on a moment in time  $t$ .

Figure 5 shows an example system. The nodes represent an instance base (a) and two depending artifacts, the edges indicate the dependencies between them. Some of these nodes may have the same owner, others not. The circle in the figure indicates the set of nodes for which an owner has the necessary permissions to make changes. We call this a set of *controllable nodes* and refer to this set as  $N_c$ .

We distinguish three types of dependencies:

- **Intra dependency** is a dependency within one node.
- **Controllable inter dependency** is a dependency from a node  $a$  to another node  $b$  where  $a \in N_c$  and  $b \in N_c$ .
- **Uncontrollable inter dependency** is a dependency from a node  $a$  to another node  $b$  where  $a \notin N_c$  and  $b \in N_c$ .

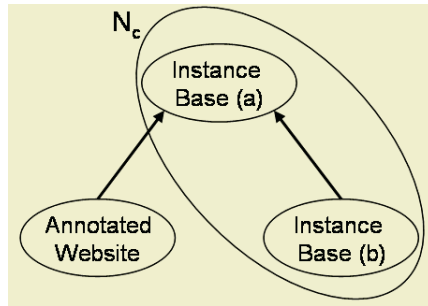


Fig. 5. Example dependency graph

Object properties, annotations, mappings between instances, etc. are forms of dependencies as they define reliance between objects. These are captured by the concept *Dependency*. A dependency exists between a source and a target instance. The concept of the source instance is called *domain* concept of the dependency, the concept of the *target* concept is called *range* concept. Furthermore, we have defined three subtypes of *Dependency*: *IntraDependency*, *ControllableInterDependency* and *UncontrollableInterDependency* referring to the distinction we introduced above. We define the set  $D$  as the set of all dependencies.

Current ontology evolution approaches provide solutions for keeping a system consistent by propagating a change to an instance to all depending artifacts [12] [13]. This means that one change may result into a chain of changes to related artifacts to avoid inconsistencies. While this may be an appropriate approach for intra and controllable inter dependencies (where one has sufficient permissions to make changes), this solution is not suitable for uncontrollable inter dependencies. In a setting like the Semantic Web, you cannot force others to update their depending artifacts to enforce consistency. Consider for example semantically annotated websites. It is not realistic to require an update of the annotations every time a change to the instance base occurs. Furthermore, it may even be not desirable to update depending artifacts. E.g. this is the case where a web page shows an image of the current prime minister that is annotated with an instance of the concept 'PrimeMinister'. When this instance evolves into an instance of for example 'Senator', it is not desirable to update the annotation or

content of the web page when the intention of the page was to show the image of the prime minister of the particular period.

Instead of propagating changes to depending artifacts, we maintain consistency without forcing third-party users to update their depending artifacts. This is done by indicating that a dependency may be state dependent, i.e. is only valid in a particular state of the instance base. To realize this, the concept ‘UncontrollableInterDependency’ is associated with an *invalidation event type*. The invalidation event type is used to specify that the dependency becomes invalid after the occurrence of the event. In other words, the dependency is only valid in the state of the instance base before such an event occurs for the target instance. Consider as example the following situation:  $m$  ‘is member of’  $g$  where  $m$  is an instance of ‘Minister’ in an instance base (b) (of figure 5),  $g$  is an instance of ‘Government’ in instance base (a) and ‘is member of’ is an UncontrollableInterDependency between these two instances (the source of the dependency is  $m$  and the target is  $g$ ). Attaching an invalidation event type to this dependency implies that the dependency refers to the state of the instance base (a) before the occurrence of the invalidation event for instance  $g$ . When no such invalidation event occurred for instance  $g$ , the dependency refers to the actual state of the instance base (a).

To simplify specifications, we have defined a default invalidation event type. The default invalidation event occurs when the target instance of the dependency is no longer an instance of the concept defined as range of the dependency. E.g. the retirement of the instance  $g$ , would trigger the default invalidation event type because  $g$  is no longer an instance of the range concept (i.e. ‘Government’).

The default invalidation event type is defined as follows:

$$\begin{aligned} \forall i \in I: \text{DefaultInvalidationEvent}(i) : \\ \text{subtypeOf}('Event') \leftarrow \\ \exists d \in D: \text{instOf}(d, \text{'Unc.InterDependency'}, ct_d) \wedge \\ \exists c \in C: \text{range}(d, c) \wedge \text{target}(d, i, ct_d) \wedge \\ \text{instOf}(i, c, ct_i-1) \wedge \neg \text{instOf}(i, c, ct_i) \end{aligned}$$

This event type specifies that there exists an uncontrollable inter dependency  $d$  and the range of the dependency is a concept  $c$ . Furthermore, an instance  $i$  is the target instance of the dependency  $d$ , but is no longer an instance of the range concept  $c$ .

Although, we specify a default behavior for uncontrollable inter dependencies, users can always refine this default setting by adding their own invalidation event types. Consider an annotated web page where there exist a dependency between page objects and instances of an instance base. (Note that annotations are a specific type of dependency as the source instance of the dependency is an instance of a HTML element.) Suppose the web page presents an annotated group picture of all ministers of the current government. Here, the default invalidation event type will not give the desired effect. When one or more ministers leave this government, the picture is no longer a correct representation of the state of the government. Therefore, the following event type should be added to the annotation as an additional invalidation event type and is defined as follows:

$$\forall i \in I: \text{InvalidationEvent}(i): \text{subTypeOf}('Event') \leftarrow \\ \text{occurredEvent}(i, 'MinisterLeavesGovernment', ct_i) \wedge \\ \exists d \in D, g \in I: \text{instOf}(d, 'Unc.InterDependency', ct_d) \wedge \\ \text{target}(i, g, ct_i-1) \wedge \text{target}(d, g, ct_d)$$

The definition checks if the event ‘MinisterleavesGovernment’ (see section 6.1) occurred for an instance  $i$  for which a dependency  $d$  exists with as target instance, the target instance of  $i$ .

## 8 Conclusion

We have presented an approach for ontology evolution on the instance level. A log of changes is maintained (by means of an evolution ontology) listing all changes applied. Third-party users can use this log to check if relevant changes occurred by specifying event types. If, after a change, instances satisfy the definitions of one of the event types, an instance of this event type is created. The event types are defined in *an event ontology* and the events itself are captured in a log of events. Instead of forcing third-party users to update their dependent artifacts to maintain consistency after a change, we have presented an event-based technique for maintaining consistency. Event types are used to invalidate dependencies and to refer to previous states of an instance base.

The advantages of our approach can be summarized as follows:

- Evolution of instances can be maintained without touching the instance base by means of the evolution ontology.
- Event types allow to filter relevant changes and to establish the semantics of changes in term of real life events. Furthermore, an event type is an abstraction mechanism that allows to reason about changes on a higher level of abstraction than possible with changes in the evolution ontology.
- Depending artifacts can be kept consistent without forcing third-party users to make updates.

## References

1. Berners Lee, T., Hendler, J., Lassila, O.: The Semantic Web: A new Form of Web Content that is Meaningful to Computers will unleash a Revolution of new Possibilities. *Scientific American*, 5(1) (2001)
2. Gruber, T.R.: A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2) (1993) 199-220
3. Klein, M., Noy, N. F.: A Component-based Framework for Ontology Evolution. *Proceedings of the Workshop on Ontologies and Distributed Systems (IJCAI '03) Acapulco Mexico* (2003)
4. Maedche, A., Motik, L., Stojanovic, L., Studer, R., Volz, R.: Ontologies for Enterprise Knowledge Mmanagement. *IEEE Intelligent System* 18(2) (2003) 26-34
5. Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., Snodgrass, R.: *Temporal databases: Theory, Design and Implementation*. Redwood City, CA: Benjamin/Cummings Pub. (1993)

6. Ozsoyoglu, G., Snodgrass, R.: Temporal and Real-time Databases: A survey. *IEEE Transactions on Knowledge and Data Engineering* 7(4) (1995) 513-532
7. Tausovitch, B.: Toward Temporal Extensions to the Entity-Relationship Model. 10th International Conference on the Entity Relationship Approach (1991) 163-179
8. Klopprogge, M., Lockeman, P.: Modeling Information Preserving Databases: Consequences of the Concept Time. Ninth International Conference on Very Large Data Bases (1983) 399-416
9. Theodoulidis, C., Loucopoulos, P., Wangler, B.: A Conceptual Modelling Formalism for Temporal Database Applications. *Information Systems* 16(4) (1991) 401-416
10. Goralwalla, I., Ozsu, M.: *An Object-Oriented Framework for Temporal Data Models*. Springer-Verlag, ABERlin Heidelberg (1998)
11. Dey, D., Barron, T., Storey, V.: A Conceptual Model for the Logical Design of Temporal Databases. *Decision Support Systems* 15 (1995) 305-321
12. Maedche, A., Motik, B., Stojanovic, L.: Managing Multiple and Distributed Ontologies on the Semantic Web. *The VLDB Journal – Special Issue on Semantic Web* 12 (2003) 286-302.
13. Maeche, A., Motik, B., Stojanovic, L., Studer, R., Volz, R.: An infrastructure for Searching, Reusing and Evolving Distributed Ontologies. Twelfth International World Wide Web Conference (WWW 2003), Budapest Hungary (2003) 51-62



# Interoperability in Meta-environments: An XMI-Based Approach

Roberto Riggio<sup>1</sup>, Domenico Ursino<sup>1</sup>, Harald Kühn<sup>2,\*</sup>,  
and Dimitris Karagiannis<sup>3</sup>

<sup>1</sup> DIMET, Università “Mediterranea” di Reggio Calabria, Via Graziella,  
Località Feo di Vito, 89060 Reggio Calabria, Italy  
roberto.riggio@gmail.com, ursino@unirc.it

<sup>2</sup> BOC Information Systems GmbH, Rabensteig 2, A-1010 Vienna, Austria  
harald.kuehn@boc-eu.com

<sup>3</sup> Institute for Knowledge and Business Engineering, University of Vienna,  
Brünnerstrasse 72, A-1210 Vienna, Austria  
dk@dke.univie.ac.at

**Abstract.** In this paper we propose an approach conceived to handle the interoperability in meta-environments. The paper first illustrates the relevance of model interoperability in the present software engineering applications; then, it presents the proposed approach, with a particular emphasis to the relevant role MOF and XMI play in it. Finally, it illustrates a prototype we have realized for verifying the applicability of the proposed approach in a real case, namely the Business Process Management domain.

## 1 Introduction

One of the most important trends that are presently characterizing the software engineering community is the larger and larger exploitation of the model engineering paradigm. Its adoption is leading to a revolution analogous to that characterizing the 80's of last century, when procedural programming has been substituted by the object-oriented paradigm.

In such a context models will play the key role; they will be exploited not only for documentation but also for software development; they will benefit of software automatic generation techniques.

In this scenario, the Object Management Group (OMG) [9] has proposed to shift from the classic Object Management Architecture (OMA) [3], characterized by an interpretative approach based on the development of complex middleware platforms such as CORBA [8], to the Model Driven Architecture (MDA) [4] characterized by a generative approach based on model transformation.

---

\* This work is partially supported by the Commission of the European Communities under the sixth framework programme (INTEROP Network of Excellence, Contract N° 508011, <<http://www.interop-noe.org>>).

One of the key issues characterizing this revolution is the necessity to move model-relevant information from one development environment to another one in a transparent and efficient way. This is even more important in case of round-trip engineering since, in this context, it is necessary to migrate models among modelling platforms in a bi-directional way.

Model reuse in environments different from those they have been realized in, has several motivations. Some of the most important ones are the following:

- a single modelling tool typically cannot be used during the whole life cycle of an information system under development, i.e. from its strategic planning to its maintenance;
- even in integrated modelling environments, the various components might be not able to show the best performance for each phase of the system life cycle;
- in the development of highly heterogeneous systems, an organization might decide to use different methods or development processes; as a consequence, a single development tool might be not capable to satisfy all requirements;
- the life time of some projects might be of several decades; it can be easily foreseen that no presently available modelling tool will be available, or at least be retro-compatible, for such a long time;
- in large projects, spread over different companies, there is only a little chance that all participants will use the same set of development tools.

All the examples illustrated above allow us to conclude that, without a good interoperability among different modelling environments, users will be forced to exploit a small set of development tools or, alternatively, to totally renounce to their modelling activity.

The MDA allows the definition of various approaches for handling model interoperability; in this paper we propose a solution based on meta-model transformation. The architecture underlying our solution is shown in figure 1 and is based on the ideas developed in [12].

The core of our proposal consists of the exploitation of a common meta-meta-model and a meta-data exchange facility. In our approach, this has been identified in the Meta Object Facility (MOF) [5, 26] and the XML Metadata Interchange (XMI) [7, 15,]. As it will be clear in the following, the exploitation of these two standards allows uniformity among involved models to be easily gained.

We shall illustrate all details of our approach in the next section. Here, we consider extremely relevant to point out that its feasibility has been extensively verified in a real application case, in particular in the Business Process Management domain. In this field, various process modelling languages exist, some of them focusing on business aspects such as ADONIS<sup>1</sup> BPMS language [17], Event-driven Process Chains [19] or UML Activity Diagrams [6], others focus on execution aspects such as Business Process Modelling Language (BPML) [2], XML Process Definition Language (XPDL) [11] or Business Process Execution Language for Web Services (BPEL4WS or BPEL) [1]. These languages are characterized by significantly heterogeneous paradigms; therefore, their interoperability is difficult to be gained. In this context our ap-

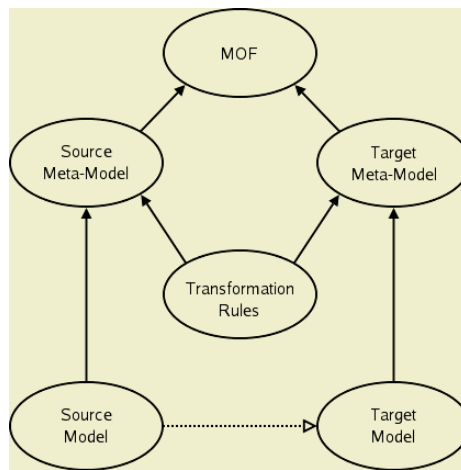
---

<sup>1</sup> ADONIS is a registered trademark of BOC GmbH. All other brands are property of the particular companies or organisations.

proach can play a key role; indeed, the capability to define mappings between the meta-models corresponding to the various languages and the MOF meta-model would automatically imply the possibility to exploit MOF as a common language for defining meta-models and XMI as a common language for meta-data exchange.

In order to verify the feasibility of our approach we have realized a prototype handling the mapping between the meta-model of ADONIS and the MOF meta-model. In our opinion, obtained results are encouraging; they are described below.

The outline of the paper is as follows: section 2 presents a general overview of our approach. Technical details are illustrated in section 3. In section 4 we describe our prototype for handling the mapping between the meta-model produced by ADONIS and the corresponding MOF meta-model. In section 5 we provide a brief overview of related work. Finally, in section 6 we draw our conclusions.



**Fig. 1.** A general approach for model transformation

## 2 Overview of the Proposed Approach

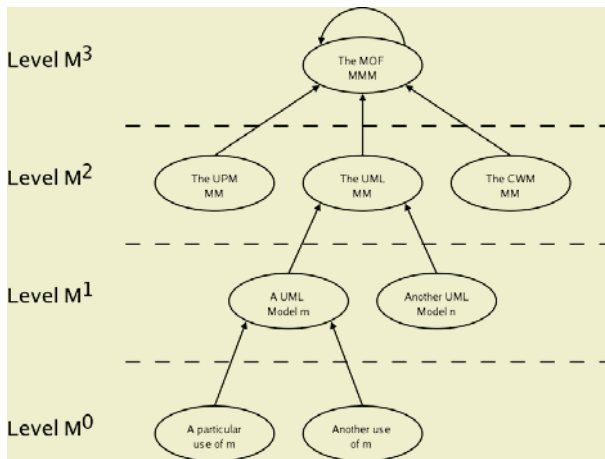
The general architecture of our approach is shown in figure 1. Both MOF and XMI play a key role in it.

Recall that XMI defines the way a generic MOF-compliant model will be represented as an XML document. For a given meta-model, each XMI-conforming implementation will produce a DTD (or an XML Schema), representing the meta-model, and an XML document, representing an instance of the given meta-model.

The specific generation rules rely on a MOF definition of the model's meta-model; therefore, a meta-model can have its models interchanged through XMI only if it is represented as an instance of the MOF meta-meta-model. It is worth pointing out that XMI works at all abstraction levels of the meta-model architecture defined by MOF. This implies that it can be used for both the object serialization and the meta-data exchange (see below).

Our architecture is a particular case of the MOF meta-data architecture. An example of the MOF architecture, tailored for the UML environment, is shown in figure 2.

- *The lowest layer*, sometimes called *original level* [18], is that originating the model and often contains run-time data. At this level XMI can be used for handling the object serialization.
- *The model layer* includes the meta-data relative to the lowest layer. Meta-data are aggregated as models. At this level XMI can be used for handling the model or the meta-data exchange among tools using the same meta-model.
- *The meta-model layer* includes the description of both the structure and the semantics of the meta-data, i.e. the meta-meta-data. The meta-meta-data are aggregated as meta-models. A meta-model is an “abstract language” for describing different kinds of data. At this level, XMI can be used for representing the model language, i.e. the meta-model.
- *The meta-meta-model layer* includes the description of both the structure and the semantics of the meta-meta-data. The use of XMI at this level allows an MOF model to be represented as an XML document.



**Fig. 2.** The MOF four-layer architecture

In the standard OMG modelling stack, the meta-meta-model (also called MOF Model) is self-defined and allows the definition of meta-models at the third layer. The UML meta-model is one of the well-known examples of meta-models; it is possible to define also other generic languages for meta-modelling. In this paper we shall focus our attention on the exploitation of XMI at the second layer of the MOF stack.

### 3 Technical Details

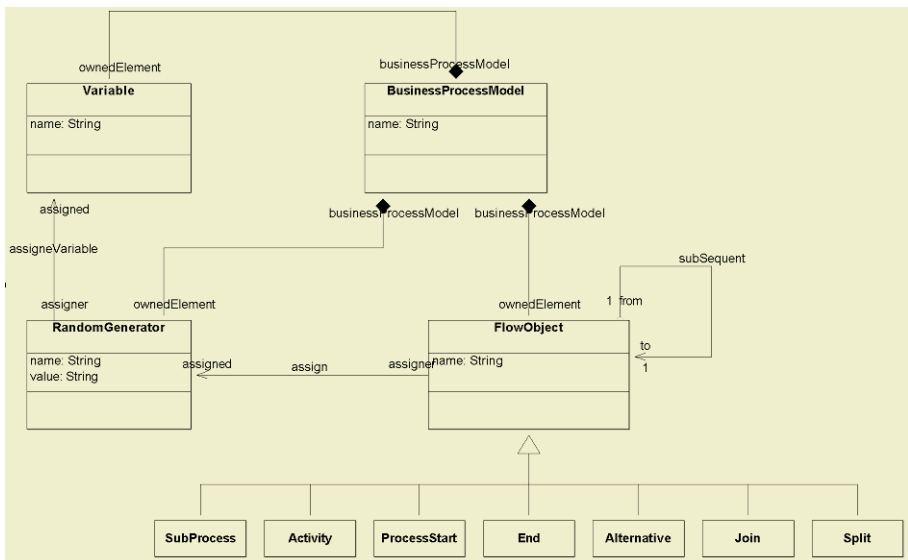
In this section we shall illustrate the proposed architecture into detail. In order to carry out such a task we shall consider, as an example, the mapping between the ADONIS

Business Process meta-model and the MOF meta-model, and the consequent translation of models produced by ADONIS into XMI-compliant documents. The considered version of the XMI specification is 1.2.

ADONIS is a business meta-modelling tool with components such as information acquisition, modelling, analysis, simulation, evaluation, process costing, documentation, staff management, and import/export [17]. Its main feature is its method independence. This means, that starting from the ADONIS meta-tool level, distinct business modelling tools with specialized meta-models can be derived. The main application area of ADONIS is Business Process Management.

Figure 3 contains a fragment of the default ADONIS Business Process meta-model. By its examination we can observe that such a meta-model consists of a composition hierarchy. This is a typical feature of most meta-models. In the hierarchy the `BusinessProcessModel` element consists of three sub-elements, namely `FlowObject`, `Variable` and `RandomGenerator`.

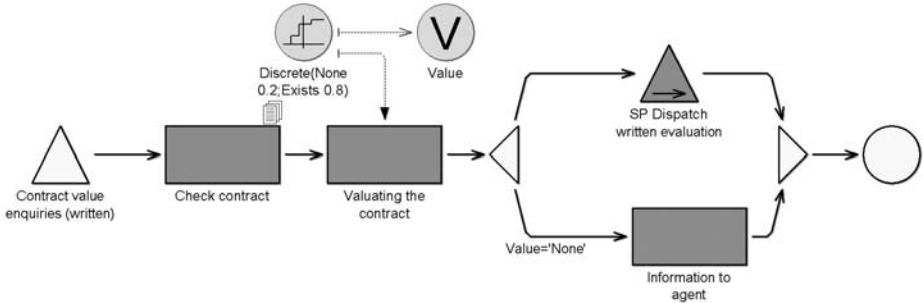
The composition is defined by means of the MOF's composite association form. An MOF composite association form is a (conceptually strong) binding among instances; it is characterized by the following properties:



**Fig. 3.** The ADONIS Business Process meta-model

- it is asymmetrical, with one end denoting the “composite” and the other one representing the “components”;
- an instance cannot be a component of more than one composite at a time, under any composite relationship;
- an instance cannot be a component of itself, of its components, of the components of its components, and so on, under any composite relationship;

- when a “composite” instance is deleted, all its components, all the components of its components, etc., are also deleted;
- an instance cannot be a component of an instance from a different package extent (composition closure rule).



**Fig. 4.** A model conforming to the ADONIS Business Process meta-model

Mapping the ADONIS Business Process meta-model into the MOF meta-model implies to apply suitable production rules to obtain an XMI-compliant XML document for each ADONIS Business Process model. A simple Business Process model, conforming to the Business Process meta-model of figure 3, is shown in figure 4. The model “BP Contract value enquiries” shows an enquiry of a customer concerning the value of his insurance contract such as a life insurance contract. After contract check, the current value of the contract is calculated. The customer will be informed in written form, which is done in the sub process “SP Dispatch written evaluation”. In parallel, if no contract value was calculated (here: likelihood of 20%), the customer agent will be informed to contact the customer.

In the following we illustrate how the production rules for obtaining an XMI-compliant XML document from an ADONIS Business Process model can be applied. For this illustration we shall consider the model of figure 4 and the corresponding meta-model of figure 3.

Production rules are applied starting from the root of the model, i.e. the unique instance of the `BusinessProcessModel` element. After the root has been considered, rules are applied throughout the model hierarchy by navigating the composition links. For each object, including the root, an element start-tag is generated; to this purpose, the name of the corresponding element in the meta-model is adopted. As an example, if we consider the root in figure 3, we obtain:

```
<BusinessProcessGraph.BusinessProcessModel
  xmi.id="od.1">
```

For each attribute of the current object, a suitable XML element is generated and the attribute is enclosed in it. The name of the element is derived from the name of the attribute, as it appears in the meta-model. As an example, the attribute name of the root in figures 3 and 4 is translated as follows:

```
<BusinessProcessGraph.BusinessProcessModel.name>
  BP Contract value enquiries
</BusinessProcessGraph.BusinessProcessModel.name>
```

Each composite association is translated in XMI by means of the XML element containment. As an example, the composite association between the elements `BusinessProcessModel` and `FlowObject` in figure 3 is translated as:

```
<BusinessProcessGraph.BusinessProcessModel.
  ownedElement>
```

As previously pointed out, after the root has been examined, the other objects of the model are taken into account. For each of them, a suitable element is written out in the corresponding XML document; such a task is carried out by following the guidelines illustrated above. As an example, the XML start-tag for representing the object `Check contract` in figure 4 (that is an instance of the element `Activity` of figure 3) is the following:

```
<BusinessProcessGraph.Activity xmi.id="obj.2">
```

Just as before, an element is created for each attribute of the object. In our example, `Check contract` is an instance of the element `FlowObject` in figure 4 and this element has an attribute called `name`; this attribute is translated as follows:

```
<BusinessProcessGraph.Activity.name>
  Check contract
</BusinessProcessGraph.Activity.name>
```

After an element and those linked to it have been examined, the end-tag corresponding to it is generated. As an example, the end-tag associated with the element `FlowObject` and the corresponding instance `Check contract` is as follows:

```
</BusinessProcessGraph.Activity>
```

Analogously, after all the elements of a composite association have been examined, an end-tag relative to it is generated. As an example, the end-tag associated with the composite association between the elements `BusinessProcessModel` and `FlowObject` is the following:

```
</BusinessProcessGraph.BusinessProcessModel
  .ownedElement>
```

Finally, as far as the simple association is concerned, its links are represented in the content of a suitable element contained in the standard `XMI.content` element. As an example, consider the cyclic association `subSequent` in figure 3, recursively linking the element `FlowObject`, and the corresponding instance in figure 4, linking `Check contract` to `Valuating the contract`; the associated XML code is the following:

```
<BusinessProcessGraph.subSequent xmi.id="con.1">
  <BusinessProcessGraph.subSequent.from>
    <BusinessProcessGraph.Activity xmi.idref="obj.2" />
  </BusinessProcessGraph.subSequent.from>
  <BusinessProcessGraph.subSequent.to>
    <BusinessProcessGraph.Activity xmi.idref="obj.3" />
```

```
</BusinessProcessGraph.subSequent.to>
</BusinessProcessGraph.subSequent>
```

In an analogous way all the other elements, attributes, composite associations and simple associations of the model can be represented within the XMI-compliant XML document.

At the end of the whole process, the end-tag of the root is generated. As far as our example is concerned, the following end-tag is written out:

```
</BusinessProcessGraph.BusinessProcessModel>
```

This closes our discussion about the translation modalities of our approach.

It is worth pointing out, that even if in our discourse we have considered the translation from ADONIS to MOF, the approach we are proposing here is general and could be applied for translating any Business Process model to MOF. For this reason, we can say that it guarantees the interoperability among different meta-models.

### 4 Prototype

In this section we describe the prototype we have realized for handling the mapping between the ADONIS Business Process meta-model and the MOF meta-model, and the consequent translation of models produced by ADONIS into XMI-compliant documents. Our prototype is characterized by two main features:

- Exporting a model produced by ADONIS into an XMI-compliant XML document.
- Importing an XMI-compliant XML document representing a model into ADONIS.

As a consequence, it allows the model interchange between ADONIS and any XMI-compliant CASE tool available in the market.

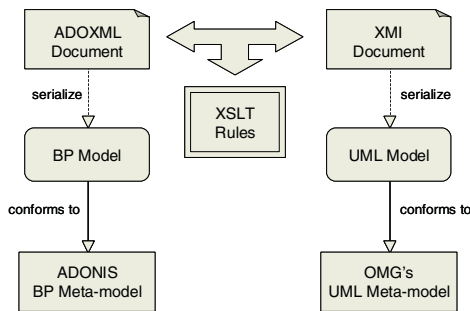


Fig. 5. The architecture of the prototype

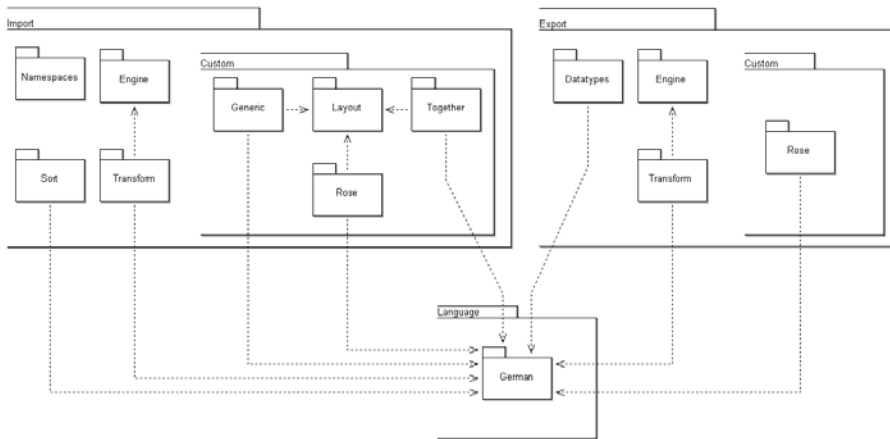
Our prototype is strongly based on the W3C family of XML-based standards conceived for handling MOF compatible meta-data. These are very variegated and allow



various transformation systems, like XSLT, to be applied to meta-data at any abstraction level. The overall framework of the prototype is shown in figure 5.

The core of the system consists of a set of XSLT templates; these are applied to the source XML document returned by ADONIS (hereafter, ADOXML document) and representing a Business Process model; they produce an XML document compliant with the XMI specifications (hereafter, XMI document). The XSLT templates can be applied also for translating an XMI document into an ADOXML one.

The structure of the XSLT templates is shown in figure 6. Three main packages can be recognized, namely: *Import*, *Export* and *Language*.



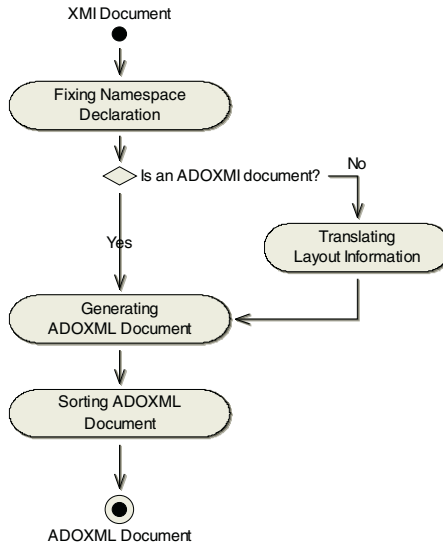
**Fig. 6.** The XSLT template structure of the prototype

The *Import* package defines the stylesheets used for translating an XMI document into an ADOXML one. This package is decomposed into the following sub-packages: (i) The *Namespace* package defines the stylesheets for fixing the namespace declarations in the source XMI document. (ii) The *Engine* package defines the stylesheets for translating UML diagrams into an intermediate ADOXML document. (iii) The *Custom* package defines the stylesheets for handling the layout information of specific CASE tools. This sheet manages also the heterogeneities regarding the representation of UML attributes and relations [16]. (iv) The *Sort* package contains the stylesheets for sorting the intermediate ADOXML document in order to produce a final XML document compliant with the ADONIS specifications.

The *Export* package defines the stylesheets used for translating an ADOXML document into an XMI one. This package is decomposed into the following sub-packages: (i) The *Datatypes* package defines the stylesheets for generating the XML document containing all the data types used in the ADOXML document. (ii) The *Engine* package defines the stylesheets for handling the UML diagrams defined into the source XMI document. (iii) The *Custom* package defines the stylesheets for tailoring the XMI document to a specific CASE tool.

The *Language* package is a support package for making our prototype available in various natural languages such as English, German etc.

The behaviour of the *Import* process is illustrated in figure 7.



**Fig. 7.** An UML Activity Diagram showing the behaviour of the *Import* process

For each activity shown in the diagram an XSLT sheet is applied to the input XML document. The various activities related to the *Import* process behave as follows:

- *Fixing Namespace Declaration.* The XSLT technology requires the explicit declaration of the namespaces used during the transformation. However, the UML namespace declaration is not consistent through different XMI implementations. Such an inconsistency precludes the stylesheet to work with a generic document. This activity aims at removing such an inconsistency.
- *Translating Layout Information.* The XMI document is examined in order to determine the exporter software. Then, a suitable XSLT sheet is applied to the document for generating an intermediate XMI document. This sheet also handles heterogeneities regarding the representation of UML attributes and relations [16].
- *Generating ADOXML Document.* An intermediate ADOXML document is generated starting from the intermediate XMI document.
- *Sorting ADOXML Document.* In this step the intermediate ADOXML document is sorted for producing the final ADOXML document compliant with the ADONIS specifications.

The behaviour of the *Export* process is illustrated in figure 8. For each activity shown in the diagram an XSLT sheet is applied to the input XML document. The various activities related to the *Export* process behave as follows:

- *Generating DataType Document.* In this step an XML document specifying data types used in the ADOXML document is generated.
- *Generating XMI Document.* During this step the output XMI document is generated starting from the input ADOXML document and the XML document containing the data type definition produced during the previous step.
- *Translating Layout Information.* Starting from the ADONIS diagram representation, a third-party CASE tool diagram representation is generated.

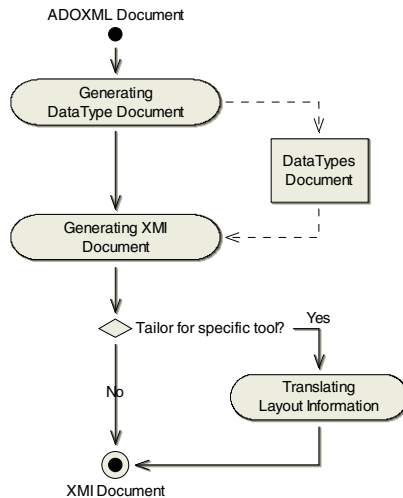


Fig. 8. An UML Activity Diagram showing the behaviour of the *Export* process

## 5 Related Work

In the following we provide a brief overview of three major categories of related work: data and systems integration, XML-based languages for Business Process Management, and model transformation approaches.

In the database management domain, issues such as schema integration [13] and data migration in federated databases [24] focus on comparable problems such as metamodel integration and model interoperability. Solutions from the systems integration domain also provide valuable input for data and meta-data integration [20, 22, 25]. Nevertheless, the richness of modelling language semantics needs additional aspects to be solved in model interoperability. One of the problems which is not covered by the aforementioned approaches is the problem of heterogeneous process flow semantics in the exchange of models in different process modelling languages.

Since the advent of XML various XML-based process description languages were published [23] such as XPD [11], BPML [2], and BPEL [1]. Additionally, XMI [7] is a candidate to be accepted as a general model and meta-model exchange format. These languages will provide valuable support for model interoperability.

But even if XMI will serve as a general model exchange facility, the semantic interoperability of models and meta-models in heterogeneous meta-environments still needs further mechanisms such as semantic model transformations to connect different modelling domains appropriately [21]. In this area, we see a strong contribution from transformation approaches in the domain of model-driven development and model engineering. In [14], a good overview of various model-to-model and model-to-code transformation approaches can be found.

## 6 Conclusions

The large heterogeneities presently characterizing Business Process Management languages makes model interoperability to play a key role in the context of meta-environments management. The presented paper gives a contribution in this setting by proposing a framework for handling interoperability among different typologies of enterprise models. Such a feature is gained by exploiting the MOF and the XMI standards.

We have developed a prototype applying the underlying ideas of the proposed framework. This guarantees the interoperability between the meta business modelling tool ADONIS and any XMI-compliant CASE tool available in the market. Obtained results are particularly encouraging; a proof of this is that an extension of the prototype realized, will be made available for ADONIS customers.

Nevertheless, we still see a number of open issues, which will guide our further research. One of these issues is the semantic interoperability of models and meta-models in different meta-environments. Even if models can be exchanged e.g. using XMI, the semantic meaning of the models and meta-models in each meta-environment may be different. Ontology may serve as an appropriate tool in this context.

Other issues for further evaluation are non-functional aspects such as performance, ease of use and security in model interoperability. E.g. currently one important performance obstacle in the practical application of the presented approach is the extensive main memory usage of the XSLT processor during transformation of large model bases.

## References

1. BPEL4WS (Business Process Execution Language for Web Services) Version 1.1 May, 5 2003. <http://www-106.ibm.com/developerworks/library/ws-bpel/>.
2. BPMI.org: Business Processing Modelling Language - Specification 1.0. <http://www.bpmi.org/bpml-spec.esp>.
3. Object Management Group: Object Management Architecture Guide. <http://doc.omg.org/ab/97-05-05>.
4. Object Management Group: MDA Guide, Version 1.0.1, June 12 2003.
5. Object Management Group: Meta Object Facility (MOF) Specification, Version 1.4, April 2002.
6. Object Management Group: OMG Unified Modeling Language Specification, Version 1.4, September 2001.
7. Object Management Group: OMG XML Metadata Interchange (XMI) Specification, Version 1.2, January 2002.

8. Object Management Group: Common Object Request Broker Architecture. [http://www.omg.org/technology/documents/formal/corba\\_iiop.htm](http://www.omg.org/technology/documents/formal/corba_iiop.htm).
9. OMG, Object Management Group. <http://www.omg.org>.
10. W3C: XSL Transformations (XSLT) Version 1.0, November 1999.
11. Workflow Management Coalition: Workflow Process Definition Interface - XML Process Definition Language. Document Number WFMC-TC-1025, Document Status-Version 1.0 Final Draft October 2002. [http://www.wfmc.org/standard/docs/TC-1025\\_10\\_xpdl\\_102502.pdf](http://www.wfmc.org/standard/docs/TC-1025_10_xpdl_102502.pdf).
12. Bézivin, J.: From Object Composition to Model Transformation with the MDA. In Proceedings of TOOLS'USA, volume IEEE TOOLS-39, Santa Barbara, California, USA, 2001.
13. Bernstein, P. A., Levy, A. Y., Pottinger, R. A.: A Vision for Management of Complex Models. Microsoft Research Technical Report MSR-TR-2000-53, Juni 2000. <ftp://ftp.research.microsoft.com/pub/tr/tr-2000-53.pdf>.
14. Czarnecki, K., Helsen, S.: Classification of Model Transformation Approaches. OOPSLA'03, Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003.
15. Grose, T.J., Doney, G.C., Brodsky, S.A.: Mastering XMI: Java Programming with XMI, XML, and UML. John Wiley Sons, 2002.
16. Jeckle, M.: OMG's XML Metadata Interchange Format XMI. In: [23], pp. 25-42.
17. Junginger, S., Kühn, H., Strobl, R., Karagiannis, D.: Ein Geschäftsprozessmanagement-Werkzeug der nächsten Generation - ADONIS: Konzeption und Anwendungen. WIRTSCHAFTSINFORMATIK, Vol. 42, No. 5, 2000, pp. 392-401.
18. Karagiannis, D., Kühn, H.: Metamodelling Platforms. Invited Paper. In: Bauknecht, K., Min Tjoa, A., Quirmayer, G. (Eds.): Proceedings of the Third International Conference on E-Commerce and Web Technologies (EC-Web2002) in conjunction with DEXA2002, Aix-en-Provence, France, 2002, LNCS 2455, p. 182.
19. Keller, G., Nüttgens, M., Scheer, A.-W.: Semantische Prozessmodellierung auf der Basis "Ereignisgesteuerter Prozessketten (EPK)". Publications of Institute of Wirtschaftsinformatik, No. 89, University of Saarbrücken, 1992.
20. Kohoutková, J.: Meta-Level Transformations in Systems Integration. In: Manolopoulos, Y., Návrát, P. (Eds.): Proceedings of the Sixth East European Conference of Advances in Databases and Information Systems (ADBIS'02), Vol. 2 Research Communications, Bratislava, Slovakia, September 2002, pp. 121-130.
21. Kühn, H., Murzek, M., Bayer, F.: Horizontal Business Process Model Interoperability using Model Transformation. In: Proceedings of the Workshop on Interoperability of Enterprise Systems (INTEREST2004) held in conjunction with ECOOP 2004 conference, Oslo, Norway, June 2004.
22. Linticum, D. S.: Enterprise Application Integration. Addison-Wesley, 2000.
23. Nüttgens, M., Mendling, J. (Eds.): Proceedings of the First Workshop on XML Interchange Formats for Business Process Management (XML4BPM2004). German Informatics Society, Marburg, Germany, March 2004.
24. Sheth, A.P., Larson, J.: Federated Database Systems for Managing Heterogeneous, Distributed and Autonomous Databases. ACM Computing Surveys, Vol. 22, No. 3, 1992.
25. Skoupý, K., Kohoutková, J., Benešovský, M., Jeffery, K.G.: HYPERMEDATA Approach: A Way to Systems Integration. In: Proceedings of Short Papers of the 3rd East European Conference on Advances in Databases and Information Systems (ADBIS'99), Maribor, Slovenia, September 1999, pp. 9-15.
26. Smith, H.: BPM and MDA: Competitors, Alternatives or Complementary. Business Process Trends, 2004.

# On the Notion of Consistency in Metadata Repository Systems

Iliia Petrov, Stefan Jablonski, and Marc Holze

Chair for Database Systems, Department of Computer Science,  
University of Erlangen-Nürnberg, Martensstrasse 3,  
Erlangen, D-91058, Germany  
{ilia.petrov, stefan.jablonski, marc.holze}@cs.fau.de  
<http://www6.informatik.uni-erlangen.de/>

**Abstract.** Repository systems handle the management of metadata and meta-models. They act as data store with a custom-defined and dynamically adaptable system catalogue. This feature finds a useful application in systems such as process engines, collaborative and information systems, CASE tools and transformation engines, in which custom-defined catalogues are rarely available due to their complex nature. In this context repositories would improve those systems' ability to adapt and allow for dynamic information discovery. Preserving the consistency of the repository data is a major challenge. Repository consistency has several aspects, the most important of which is structural consistency. It is insufficiently specified in the metadata and repository standards, and is incompletely implemented in existing systems. In this paper we propose a novel approach to enforcing structural consistency in MOF-based repositories. We describe its implementation in iRM/RMS - a prototypical OMG MOF-based repository system [35]. We show how this algorithm overcomes the deficiencies of the existing approaches and products.

## 1 Introduction

Repository systems are “shared databases about engineered artifacts” [4]. They facilitate integration among various tools and applications, and are therefore central to an enterprise. Loosely speaking repository systems resemble data stores with a new and distinguishing feature – a customizable system catalogue. Metadata are data which refer to other data; they describe (a) certain aspects of the way the data are structured (structural metadata); (b) auxiliary or system-specific properties of the data (descriptive metadata). Metadata repositories are systems for handling metadata (some handle also applications' data). The organization of repository metadata (metadata architecture) is defined in [23, 15, 7, 11, 25]. It exhibits a typical layered, multi-level structure. Preserving consistency between the different layers (Table 1) is a major challenge specific to repositories. The relationship between artifacts on two adjacent levels is the type-instance relationship, i.e. definitions on a level are instances of definitions on the next higher level. While in theory this progression can be continued infinitely, in practice it is limited to four layers due to the self-description phenomenon.

A database system contains layers  $M_0$  through  $M_2$ , where  $M_2$  is immutable. To provide a custom-defined and extensible system catalogue repository systems utilize an additional layer. Therefore the layer  $M_3$  is introduced allowing for custom-defined  $M_2$ . This however entails the specific problem of consistency between adjacent layers.

**Table 1.** Layers in OMG MOF Metadata Architecture

Layer	Name	Description
$M_3$	Meta-meta-model (MOF)	A standardized language, in terms of which definitions of underlying metamodels are expressed.
$M_2$	Meta-model	Language for defining the structure (syntax) of a whole set of application model definitions. Structural definitions may be extended with the semantics of application domain definitions.
$M_1$	Model	Application Model (Application classes, Table definitions etc.). Alternative term is “information model” [4].
$M_0$	Data	Instance Data (e.g. objects, records)

Like many other repository and meta-model standards, OMG MOF (Table 1) does not explicitly target the lowest layer containing application instance data. OMG MOF [23] is a meta-meta-model standard. The different layers it defines are shown in Table 1. Self-description here is defined by the fact that the meta-meta-model (e.g. MOF) is an instance of itself, i.e.  $M_3$ -layer definitions are instances of themselves.

Consistency in repository systems has several aspects:

- Operational consistency – deals with the interaction between repository applications and the RMS (see Section 3) and is closely related to the notion of repository transactions. There are two sub-aspects: concurrent multi-client access; and cooperative atomicity [32], i.e. atomic, structurally consistent repository update operations spanning multiple elementary API operations.
- Metadata integrity – comprises the notions of well-formedness and structural integrity. It must be automatically enforced by the repository management system. Well-formedness ensures the syntactical correctness of the model definitions within a meta-layer. Structural integrity (structural consistency [23]) guarantees the conformance of objects on one level to type definitions on the adjacent higher meta-level. Structural integrity results from the strict enforcement of the type-instance relationship across meta-layers. Without structural integrity, repository applications might create or modify metadata artifacts on  $M_{n-1}$  inconsistent with respect to their meta-classes on  $M_n$ . For example, an application may read the value of an attribute of an object whose meta-object does not exist and is therefore invalid. Structural integrity violations may occur naturally in repository systems since they allow for dynamic modification of  $M_2$ ,  $M_1$  and  $M_0$  at run time. Other systems do not face this kind of issues because they assume that the catalogue is static at run time. More examples are discussed in detail in Section 4.

MOF provides several mechanisms for controlling metadata integrity. However none of them describes structural integrity, i.e. propagation of changes to underlying

layers when instances exist, and how it can be implemented. Firstly, MOF defines a set of MOF Model constraints. Secondly, MOF defines a set of closure rules and computational semantics for the abstract mapping, and JMI defines computational semantics for the Java mapping. Thirdly, MOF provides the MOF constraint model element for expressing domain rules. Last but not least, MOF (and JMI) defines a set of repository interfaces. All of the above contribute to well-formedness. The JMI computational semantics defines the so called “lifecycle semantics” for different model elements, describing create, delete or update operations when no instances exist.

This paper discusses various aspects of repository consistency. Its major contribution is the proposed approach to enforcing consistency in OMG MOF-based repositories. Here its implementation [35] is also discussed— the iRM/RMS module of the iRM project [33]. In addition, this paper dwells on some run-time aspects of consistency, which are not touched upon in the current MOF specification version 1.4 [23] such as repository transactions and changes to the MOF reflective package facilitating structural integrity.

The paper is organized as follows: the next section describes the related work; Section 0 describes the architecture of iRM/RMS outlining its key modules and their functionality. Section 0 presents an example motivating the need for consistency in repository systems. It discusses what actions must be taken upon different kinds of repository data modifications and motivates the need for transactional support. Section 0 describes how iRM/RMS implements repository transaction. Section 0 is dedicated to metadata integrity, defining the notions of structural integrity and well-formedness. It also presents the proposed structural integrity algorithm and provides a performance evaluation as well as some considerations. Section 0 draws conclusions from the work described in this paper.

## 2 Related Work

Repository systems (Unisys Universal Repository Manager [30], CA Platinum Repository, ASG Rochade etc.), metadata repositories (IBM Repository Manager/MVS, IBM AD/CycleManager, Digital CDD Cohesion etc.), and data dictionaries (IBM DB/DC DataDictionary, DataCatalogue etc.), are closely related terms for metadata management systems, among which a significant overlap exists. A key difference between repository systems and the rest is that repository systems store both data ( $M_0$ ) and metadata. Data dictionaries store primarily structural metadata (type and schema definitions), whereas metadata repositories can handle both structural and descriptive metadata.

There are a number of metadata repository related standards: IRDS [15], PCTE [31], CDIF [6,7], MOF [23]. The ISO/IEC IRDS framework [15] defines a set of related standards [16, 17] defining a Data dictionary system (termed information resource dictionary) operating as a central point of control for a whole enterprise.

MOF (Meta Object Facility) [23] is a standardized, technology independent meta-meta model from OMG. It is gaining industry acceptance with initiatives such as the Model-Driven-Architecture, and in the fields of UML (UML Metamodel, UML Profiles) or data warehousing (OMG Common Warehouse Metamodel). The OMG MOF standard defines an abstract meta-meta model and mappings to a generic OO lan-



guage – CORBA IDL, OMG XMI. JMI (Java Metadata Interface) is a standardized MOF to Java mapping. MOF relies on XMI [24] for exchange (export/import) of meta-models. In addition, MOF defines two kinds of Meta Object Protocols (MOP): a general and completely reflective one in terms of the MOF Reflective API (Package); and a set of generated interfaces, custom-tailored to a concrete meta-model.

There are a number of MOF-based repository implementations: DSTC dMOF [10], MDR [22], CIM (JMI reference implementation) [29], Adaptive Repository Enterprise Edition [1], Unisys Universal Repository Manager [30].

Microsoft Repository [3] is another repository product. It supports only limited consistency (between  $M_0$  and  $M_1$ ,  $M_2$  is static). It is currently (Version 3) available as Metadata Services for the Microsoft SQL Server 2000. Microsoft repository implements a standard meta-model – MDC Open Information Model.

The idea of supporting consistency across different meta-levels is not repository system specific - it emanates from the field of computational reflection [21] and Meta Object Protocols [19]. Metadata integrity is a key characteristic of reflective systems [5] supporting different Meta Object Protocols: OpenC++ [8], MPC++ [14], SOM [26]. A number of languages contain built-in reflective facilities supporting MOP with high levels of intercession: CLOS [18], SmallTalk [12], Schema, Lisp etc. All of the above MOPs support intercession, which is the kind of reflection most relevant in the context of metadata integrity. Additional introspective MOPs are also available, e.g. Java Reflection API or RTTI in C++.

To recapitulate – the concept of repository consistency is not new. But it is insufficiently specified in MOF and the existing MOF repositories do not support it fully. The related concept of intercession has a long history and various implementations in reflective systems.

### 3 Architecture of iRM/RMS

The architecture of a repository system and the tasks its modules perform are defined in [3, 4, 7]. The logical architecture of the iRM/RMS module (Fig. 1) will be briefly described in this section. It provides useful insights as to how the different modules implementing algorithms discussed here interact. The architecture comprises: a repository client, which is a generic library used to build repository applications on top of it; Repository Management System (repository manager [4]), handling the metadata and providing repository clients with various services; a well defined RMS interface (repository API); persistent data store; and import/export utilities. The iRM/RMS API is the API of the repository management system. It is based on the JMI Reflective API [27] and extends it to handle the  $M_0$  data. The Metadata Manager handles the repository metadata organizing it into layers. It implements the JMI Reflective API. The Lock Manager provides isolation by preventing multiple repository clients from modifying the same pieces of repository data concurrently by employing a locking mechanism. The Consistency Manager enforces metadata integrity upon a series of metadata modification operations. The Data Store Manager handles the persistence of the metadata and the  $M_0$  data, both of which are stored in separate data stores due to the significant difference of the data properties. Currently iRM/RMS uses Oracle 9i databases as data and metadata stores.

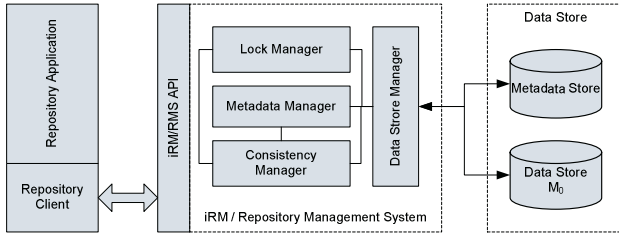


Fig. 1. Conceptual Architecture of the iRM/RMS

## 4 Introductory Example

In this section we present an example motivating the need for repository consistency. Fig. 2 shows two meta-models ( $M_2$ : RDB and  $M_2$ :RDBEx) and two models ( $M_1$ :Schema1 and  $M_1$ :Schema2). For simplicity reasons we focus only on the interaction between  $M_1$  and  $M_2$ , but the example may be easily extended to include  $M_0$ . Without structural integrity, for example, the object  $M_1$ :Emp1obj (Fig. 2) may be moved from Schema1 to Schema2, which is clearly an invalid operation since Schema2 is an instance of another package. If the application then deletes the  $M_2$ :RDBEx package and consequently all contained classes and their instances, then  $M_1$ :Emp1obj must be deleted too, which would make links ( $M_2$ :hasAttr instances) to  $M_1$ :Col1obj and  $M_1$ :Col2obj invalid. As a result Schema1 will not conform to the  $M_2$ :RDB model because  $M_1$ :Col1obj and  $M_1$ :Col2obj are not associated with instance of table anymore.

Consider the meta-classes  $M_2$ :Table and  $M_2$ :Column, the containment relationship “has” (Fig. 2) and the multiplicity of its role (its association end) “columns”, which model a rudimentary definition of a database table.

Case 1: The containment relationship “hasAttr” implies that all instances of a  $M_2$ :Table (i.e. type definitions or  $M_1$  instances) must have an element instance of “ $M_2$ :Column” as part of their composite structure. Therefore the RMS must disallow the creation of an instance  $M_2$ :Column, not associated with an instance of  $M_2$ :Table. This entails two specific problems: (a) the RMS must automatically check whether any existing instance of Column is associated with instance of table when enforcing structural integrity; in case of failure the modifications must be undone. (b) the RMS API object model provides different constructs for creating the different model elements one-at-a-time. For example, first Emp1obj instance will be created, then the Col1obj and the link between them. To solve the above problems we need the concept of repository transactions. Demarcating which RMS operations belong together and must be executed in an atomic manner as a logical unit (problem (b)) is a classical transaction processing problem. Structural integrity (problem (a)) can be enforced in deferred manner, after the end of the transactions. This is the reason why repository transactions are needed.

Case 2: Modification of models must be handled properly; the respective changes must be propagated on all underlying levels. Deleting the generalization relationship between Table and TableEx, for example, would mean that TableEx will not inherit

the attributes Name and colCnt. Enforcing structural integrity must result in the removal of the instances of the attributes Name and colCnt from all instances of TableEx (e.g. Dept1obj).

Case 3: Deleting just the RDBEx package, would force the RMS to automatically delete all of its contained elements (TableEx, Trigger and the association triggers) and their instances when enforcing structural integrity.

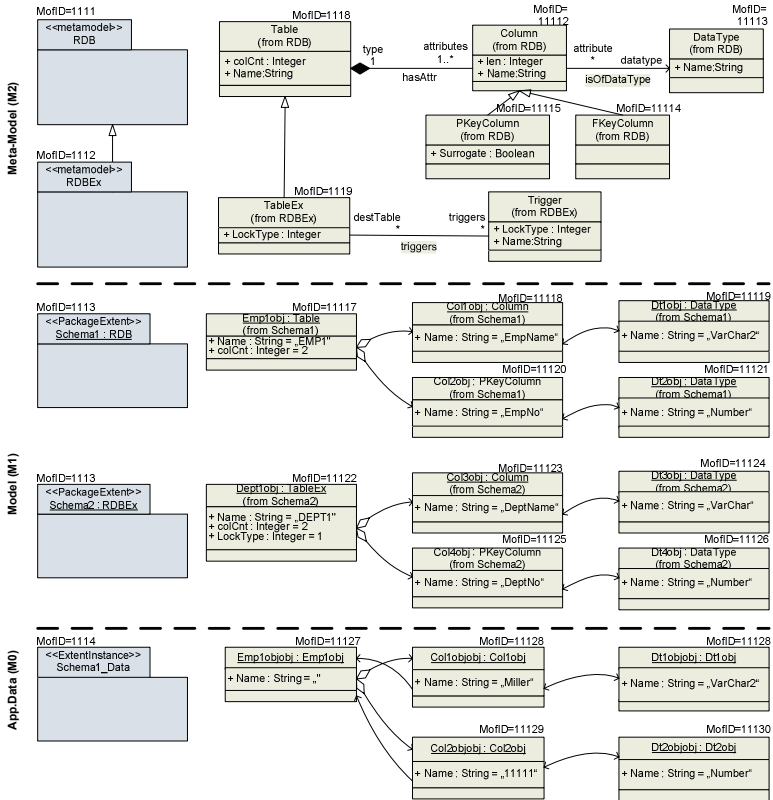


Fig. 2. Example of conformance with the metamodel

Existing OMG-MOF repository implementations either completely ignore structural integrity or implement it only partially. The case of adapting existing instances after the meta-model has been changed (or revoking the operation) is handled in none of these products. In the following sections we will describe the structural integrity mechanism implemented in iRM/RMS.

## 5 Implementing Operational Consistency

As motivated in Section 0, from the standpoint of a repository client consistency in repository systems has multiple aspects, most of which are related to providing trans-

actional support. The concept of repository transactions was first described in [4]. The primary aspects are concurrent multi-client access, and cooperative atomicity. [32] defines cooperative atomicity as “a group of concurrent activities externally viewed as a single action and synchronized with respect to the group as a whole”.

iRM/RMS repository management system is designed to handle concurrent multi-client operations. To ensure isolation the iRM repository API provides a locking mechanism based on long exclusive (X-locks) and short shared (S-) locks. The employed locking mechanism is 2PL compatible. It extends the traditional locking [14, 28] and especially the multi-granularity locking in OODB [32]. We introduce what we call instance lattice to accommodate locks on objects on the multiple meta-levels ( $M_0 \dots M_2$ ).

**Table 2.** Extended locking table / instance lattice

		Requested		
		X	S	
↖	X	-	-	$M_2$
	S	-	+	
↖	X	-	-	$M_1$
	S	-	+	
↖	X	-	-	$M_0$
	S	-	+	

Present

In principle, changes to a meta-level  $M_n$  ( $0 \leq n < 3$ ) affect the structural integrity of all instance models on all the underlying levels. In other words, once a repository client makes changes on  $M_2$  level model, all  $M_1$  instance models and the respective  $M_0$  data, must be altered accordingly by the RMS. Hence the following rule: X-lock on  $M_n$ , ( $0 \leq n < 3$ ) will be set if and only if an X-lock can be set on all instance models (and data) on all underlying meta-levels  $M_{n-p}$ , ( $0 \leq p \leq n$ ) (see Table 2). X- and S-locks would be set unless a lock is set on all instance models on all of the underlying levels (Table 2) – hence the name instance lattice. Within a single level the traditional locking rules [13] hold. More details may be found in [34].

## 6 Managing Metadata Integrity

Metadata integrity is a vital property of the repository metadata architecture. As pointed out in Section 1 metadata integrity has multiple aspects, enforced by different modules of the RMS at different times. Metadata integrity is subdivided into well-formedness and structural integrity. While well-formedness guarantees syntactical correctness of the artifacts and models, structural integrity must ensure among other things that definitions on level  $M_n$ ,  $0 < n < 3$ , conform to the model on the next higher meta-layer. In the next sections we describe these aspects in detail.

## 6.1 Well-Formedness of the Metamodels

Well-formedness guarantees the syntactical correctness of the artifacts defined within a single meta-layer at the time they are created. For example, well-formedness ensures whether the created packages, classes, associations etc. have the proper syntax; whether an association is not created with just one association end; or whether an attribute is not defined without a data type. Well-formedness can be enforced ad hoc without reasoning about the next higher meta-layer.

“Syntactical correctness”, enforced by well-formedness, implies a set of fixed rules predetermined by the “hard-wired” MOF abstract syntax. Therefore they are formulated as “constraints” (MOF Model Constraints) in the MOF specification. Some of the syntactical rules have immediate evaluation policy, i.e. after the end of a single modification operation. Satisfying these rules determines the minimum level of repository consistency. In iRM/RMS well-formedness of this type is enforced by the Metadata Manager (Fig. 1) alone. Some examples are given below:

- Enforce the properties of generalization hierarchies and containment hierarchies. For example, no class or package can be defined as super types of themselves; no name conflicts with super-type elements are allowed.
- Check whether containment rules are satisfied. For instance, operations may contain only parameters, constraints and tags; or operations are allowed to have at most one parameter marked as return.
- Check whether miscellaneous properties are properly set. Such properties are, for instance: frozen, root or leaf, singleton for abstract classes. For example, creating sub-classes of a newly created class whose attribute “isLeaf” is set to true should be disallowed.

## 6.2 Structural Integrity

Structural integrity is the major aspect of metadata integrity and a crucial characteristic of the layered repository metadata architecture. Structural integrity ensures cross-level integrity, i.e. the structure of the objects on a layer  $M_n$  conforms to the type definitions on the upper layer  $M_{n+1}$ ,  $0 \leq n < 3$ . It concerns changes of M2 or M1 level artifacts, whose instance objects exist on underlying levels. In iRM/RMS structural integrity is enforced automatically by the Consistency Manager (Fig. 1) in deferred manner. Therefore operational consistency (Section 0) is a prerequisite.

Structural integrity is expressed in terms of conformity to the respective meta-model ( $M_{n+1}$ ) and the so-called structural constraints. Structural constraints (not to be mixed with MOF Model Constraints) are conditions expressed in any constraint language such as OCL.

In section 0 we already introduced some examples as to what should happen when  $M_2$  and  $M_1$  models are modified. Such considerations emanate from topics such as architecture of a meta-language [11, 25] and meta-modeling [9]. Structural constraints, which result from the structural constraints of different MOF model elements, are expressed as follows:

*Binary associations* express a generic kind of relationship between instances of the connected model elements. Every association comprises two association ends, which are of the type of the connected model elements. Creating new association entails the

creation of new objects and their respective proxy. If an association is deleted then the association ends and all respective instances (called MOF links) are to be deleted, too. Changing the type of an association end requires the new type to be a super-type of the old one. Type incompatibility leads to repository transaction rollback.

*Containment hierarchy and Generalization hierarchy* are instrumental to the organization of the meta-models and the architecture of a meta-language [13 (p. 23)]. While a generalization hierarchy defines model elements, a containment hierarchy defines nesting and containment rules. Changing the hierarchy means changing the structure of the lower level artifacts. Restructuring existing instance artifacts is complex and cannot be performed in every case. Therefore an attempt to change a hierarchy when instance artifacts exist causes a repository transaction rollback. Such an action is allowed only if no instances exist.

*Multiplicities of association ends or attributes* define the number of instances of type the association end's type (or attribute type) included in the instance artifact and whether they are optional or mandatory. Changing a multiplicity, therefore, results in checking optionality or the lower-upper bound conditions. The repository transaction is aborted if any of these conditions is violated.

*Attributes and attribute types.* Attributes of a meta-class serve to define static properties or properties of the instance artifacts, e.g. name, count etc. Composites, i.e. attributes of type MOF classes, or aggregation relationships result in nested structures in the instance artifact. Attributes can be added or deleted. Adding a new attribute results in creating new instance objects and initializing them to the default value (NULL, if none is defined). Adding new static attribute requires that a class definition is changed. The deletion of an instance removes the instance of the attribute associated with all instances of a class. Deletion of a static attribute causes the class definition to change. If an attribute type changes the compatibility between the new and the old data type must be tested. Compatibility of primitive data types is defined in MOF and JMI. If the attribute is of type class then the new attribute type can only be a super-class of the old one. Type incompatibility leads to transaction rollback.

*Classes* can be created, deleted or altered. Creating a new class yields the creation of a new class object and the respective proxy. Deleting a class means deleting its proxy and all its instances. If the class is a type of an association end, the association end is deleted as a result of the class deletion, which would subsequently lead to the deletion of the whole association. The deletion of the class must fail if it violates a containment or generalization hierarchy.

*Packages* are generic containers for module elements. Creating a package means creating its object and its type definition (package proxy). Deleting a package means deleting its proxy and all contained elements, their proxies and instances.

An additional consideration results from the fact that a repository is a reflective system [23, 5]. Every repository object must have a respective meta-object. In case of  $M_0$  data – every  $M_0$  data element must have a respective  $M_1$  metadata element. If the meta-objects are missing or changed then the respective objects and all instance objects must be deleted or altered, too. The complementary rule states that upon insertion of a new element to a meta-definition (e.g. an attribute to a class) all instance objects must be extended with values initialized to the default value of the type, and the respective  $M_0$  data elements must be added. Type definitions in repository systems must be dynamic. Some implementation languages, e.g. Java or C++, do not support

dynamically changeable class definitions in contrast to other languages such as Smalltalk or Eiffel. JMI utilizes the concept of proxies to handle repository managed types and instances in repository applications. Upon enforcing structural integrity all proxies must be rebuilt to make the repository application coherent with the repository data. If meta-objects of package, class and association exist without a proxy then their proxies are automatically created.

### 6.3 Algorithm for Enforcing Structural Integrity

In this section we define (in pseudo-code) the algorithm for enforcing structural integrity implemented by Consistency Manager. For reasons of simplicity we have skipped parts of the algorithm handling elements such as operations or complex data types (e.g. structures or enumerations).

Algorithm 1 triggers structural integrity check for  $M_2$  models, eventually triggering Algorithm 2. Input data is a reference to the package proxy of the respective model. In Algorithm 1 all elements contained in the respective package are enumerated by traversing the containment hierarchy (line 1) and then multiplicities of all attributes or references are checked (3). All deferred well-formedness constraints are enforced (4). Eventually all  $M_1$  instance models are enumerated and for each one a structural integrity check (6) is carried out. Step (6) triggers Algorithm 2.

---

#### Algorithm 1: Multiplicity Check

---

```
performTACommitChecks( MojPackage rootPkg ){
1:   for all element in contents( rootPkg ) do
2:     for all feature in allStructuralFeatures( metaObject( element ) ) do
3:       checkMutiplicity( feature )
         end for
     end for
4:   checkDeferredEvaluationConstraints(rootPkg )
5:   checkStructuralIntegrity( getM1PackageExtent(rootPkg) )
6:   createMissingProxyObjects(rootPkg )
}
```

---

#### Algorithm 2: Structural Integrity

---

```
checkStructuralIntegrity( RefPackage pkg ){
1: if not hasValidMetaObject( pkg ) then
2:   delete( pkg )
3:   return
   end if
4: if containmentOrGeneralizationHierarchy-
   Changed( pkg ) then
5:   throw InconsistencyException("Package has
   been moved to another Package or is not
   inherited any more.");
   end if
6: for all package in refPackagesInPackage(pkg) do
7:   checkStructuralIntegrity( package )
   end for
8: for all class in refClassesInPackage( pkg ) do
9:   if not hasValidMetaObject( class ) then
10:    delete( class )
11:    continue with next iteration
   end if
12: if containmentOrGeneralizationHierarchy-
   Changed( class ) then
13:   throw InconsistencyException("Class has
   been moved to another Package or is not
   inherited any more.");
   end if
14: for all instance in allOfClass( class ) do
15:   for all instanceAttr in allAttributes(instance)
   do
16:     if hasBeenDeleted( instanceAttr ) then
17:       removeFromInstance( instanceAttr )
   end if
18:   if valueCompatibleWithType( instanceAttr )
   then
19:     convertValue( instanceAttr )
   else
```

```

20:   throw InconsistencyException("Attribute
    value's type has changed and is not
    compatible.");
    end if
  end for
end for
21: for all classAttr in allAttributes( class ) do
22:   if hasBeenDeleted( classAttr ) then
23:     removeFromClass ( classAttr )
    end if
24:   if valueCompatibleWithType( classAttr ) then
25:     convertValue( classAttr )
    else
26:     throw InconsistencyException("Attribute
    value's type has changed and is not
    compatible.");
    end if
  end for
27: for all mmAttr in attributesDefinedFor-
    Metaobject( class ) do
28:   if isInstanceScoped( mmAttr ) then
29:     for all instance in allOfClass( class ) do
30:       if not hasRepresentation( mmAttr,
        instanc
31:         addToInstance( mmAttr, instance )
        end if
      end for
    else
32:     if not hasRepresentation( mmAttr, class )
      then
33:       addToClass( mmAttr, class )
    end if
  }
}
end for
end for
34: for all assoc in refAssociationsInPackage(pkg)
    do
35:   if not hasValidMetaobject( assoc ) then
36:     delete( assoc )
37:     continue with next iteration
    end if
38:   if containmentOrGeneralizationHierarchy-
    Changed( assoc ) then
39:     throw InconsistencyException("Association
    has been moved to another Package or is
    not inherited any more.");
    end if
40:   for all link in allLinks( assoc ) do
41:     if not linkEndsMatchAssociationEnd-
    Type(assoc, link) then
42:       throw InconsistencyException("
        LinkEnd does not match
        AssociationEndType");
      end if
    end for
  }
}
Type(assoc, link) then
42:   throw InconsistencyException("
    LinkEnd does not match
    AssociationEndType");
  end if
end for
end for
}

```

---

## 6.4 Evaluation

In this section we briefly discuss the performance of iRM/RMS with respect to the consistency implementation. All presented results reflect the combined effect of enforced operational consistency and metadata integrity.

We performed extensive tests to prove experimentally the validity of the proposed algorithm. The tests were constructed to handle various cases (some of which were described in Section 0). The algorithm performed successfully in any of the following cases: changes to the abstraction hierarchies (aggregation and generalization hierarchy); creation, deletion and modification of classes; creation and deletion of attributes; attribute multiplicity checking; changing the data type of an attribute; validity of references; association end multiplicity and type checking; deletion of packages (nested and sub-packages). The experimental results cover fully the required computational semantics described in JMI. In addition the experiments showed that the proposed consistency algorithm behaves correctly.

The performance tests were performed on a Pentium 3, 1.13GHz computer with 512 MB RAM. All measured times are in milliseconds. All tests were performed without data store support. The reason for this is twofold: to discard the influence of issues such as distribution; to avoid the influence of the underlying database system and type of storage schema.



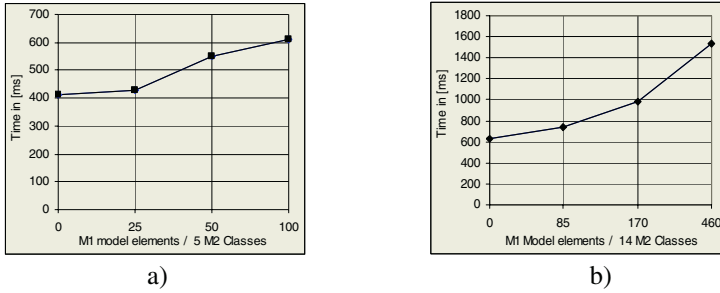


Fig. 3. Creation of M2 and M1 models

The first group of measurements (Fig. 3) shows the performance of iRM/RMS when creating M<sub>2</sub> and M<sub>1</sub> models of different size. Fig. 3.a depicts the performance in the case of a small M<sub>2</sub> model with 5 Classes (total of 8 elements), while varying the number of structurally conform M<sub>1</sub> model instances and checking for consistency. Fig. 3.b depicts the system performance in the case of larger M<sub>2</sub> and M<sub>1</sub> models. An approximately linear dependency between the consistency enforcement performance and the number of checked elements can be seen, as it may be expected.

Fig. 4 shows the system performance when carrying out modification operations on the M<sub>2</sub> model with existing M<sub>1</sub> model instances. Fig. 4.a shows the performance of the structural integrity algorithm, when modifying M<sub>2</sub> model elements, with existing 40 M<sub>1</sub> models with a total of 750 elements. The graph in Fig. 4.a shows that the structural integrity algorithm exhibits acceptable performance on this relatively small set of data.

The most expensive operations are changing an attribute or operation in a M<sub>2</sub> class, which is root of a generalization hierarchy. Fig. 4.b shows the performance of the structural integrity algorithm on the same data set, when the M<sub>2</sub> model is modified, and is not inconsistent with the existing data. Fig. 4.b shows a slight exponential curve. Detecting conflicts with changed attribute types or operation parameter types in containment or generalization hierarchies is the most time-consuming case.

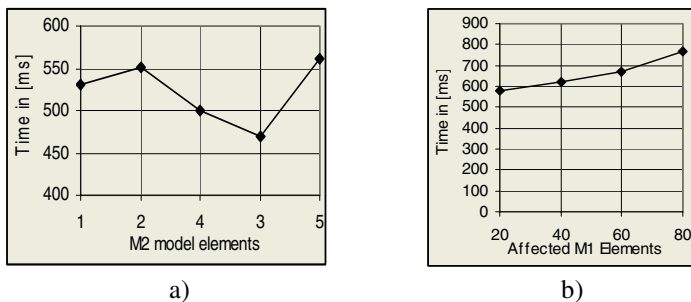


Fig. 4. Modification of M2 models

The test results show that the iRM/RMS consistency implementation exhibits acceptable performance, on the average, based on the small set of measurements and without persisting the repository data in a data store. The results show that the realization is suitable for the typical repository use, which involves mainly read and create operations on repository data on different meta-layers. The results illustrate that validation or updates on repository data involving structural integrity exhibit acceptable performance, which could be improved. iRM/RMS fails to show good performance when enforcing structural consistency on incorrect  $M_1$  or  $M_2$  models. In a realistic environment however significant performance penalty is incurred, predominantly due to the performance of the underlying data store.

Authors of the MOF specification [23] hint at the existence of different degrees of consistency although the specification does not specify any in particular. We distinguish five categories – the lowest denoted by “0”, and the highest being “4” (see Table 3). Each degree requires that all lower number degrees are satisfied.

**Table 3.** Degrees of metadata integrity

Degree	Description
0	Well-formedness - immediate
1	Transactional support (operational consistency)
2	Well-formedness – deferred
3	Structural integrity
4	Support for MOF Constraints (rules) expressed in a constraint language, e.g. OCL

Only iRM/RMS covers degree 3, whereas the majority of existing implementations cover degree 2. iRM/RMS does not have an OCL support therefore it cannot evaluate dynamically MOF Constraints. This disadvantage prevents metamodel designers from specifying a whole class of structural constraints reflecting the specifics of an application domain on  $M_2$ .

## 7 Conclusions

In this paper we presented an approach to providing consistency in OMG MOF-based repository systems, which is implemented in iRM/RMS. Repository consistency, has many facets and structural integrity is a major one. It is insufficiently specified and incompletely implemented in existing MOF-based repository systems.

We showed that the concept of repository transactions is needed for enforcing structural integrity in a deferred manner. To reflect the specifics of the repository systems the locking mechanisms known from OODBMS need to be extended. We proposed such an extension.

In this paper we also describe the way structural consistency was implemented in iRM/RMS. We define a general algorithm for enforcing structural integrity, which is the major contribution of the paper.

## References

- [1] Adaptive Ltd. Adaptive Enterprise Repository (White Paper). May 2002
- [2] Bernstein, P. Repositories and Object-Oriented Databases. Proceedings of BTW '97, Springer, March 1997
- [3] Bernstein, P., T. Bergstraesser, J. Carlson, S. Pal, P. Sanders, D. Shutt. Microsoft Repository Version 2 and the Open Information Model. Information Systems 24(2), 1999, pp. 71-98.
- [4] Bernstein, P., U. Dayal: An overview of repository technology. Proceedings of the 24th VLDB Conference Santiago Chile, 1998
- [5] Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture - A System of Patterns. John Wiley & Sons. August 1996.
- [6] CDIF - Integrated Meta-Model EIA CDIF documents EIA/IS-111 EIA/IS-112, EIA/IS-114, EIA/IS-115
- [7] CDIF CASE Data Interchange Format - Overview EIA CDIF document EIA/IS106
- [8] Chiba, S. A metaobject protocol for C++. In 10th Annual Conference on Object-oriented Programming Systems, Languages and Applications, volume 30 of ACM SIGPLAN Notices, pages 285-299, October 1995
- [9] Clark T., A. Evans, S. Kent: Engineering Modelling Languages: A Precise Meta-Modelling Approach. FASE. pp 159-173. 2002
- [10] Cooperative Research Centre for Distributed Systems Technology (DSTC). dMOF Version 1.1. User guide. 2000.
- [11] Erich, O. Repository Systems Teil 1: Mehrstufigkeit und Entwicklungsumgebung" and "Repository Systems Teil 2: Aufbau und Betrieb eines Entwicklungsrepositoriums" . Informatik-Spektrum, Abstract Volume 22 Issue 4 (1999) pp 235-251 and Abstract Volume 22 Issue 5 (1999) pp 351-363
- [12] Foote, B. R. E. Johnson. Reflective Facilities in Smalltalk-80. SIGPLAN Notices. 1989
- [13] Gray, J., A. Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, 1994
- [14] Ishikawa, Y., A.Hori, M.Sato, M.Matsuda, J.Nolte, H.Tezuka, H.Konaka, M.Maeda, K.Kubota. Design and Implementation of Metalevel Architecture in C++, MPC++ Approach. Reflection '96 Conference, April, 1996
- [15] ISO/IEC 10027:1990 Information Technology - Information Resource Dictionary System Framework 1990- 06-15
- [16] ISO/IEC 10728, Information Technology - Information Resource Dictionary System (IRDS) Services Interface, April 1993
- [17] ISO/IEC 13238-3, Information Technology - Data Management - Part 3: IRDS export/import facility, Dec. 1998
- [18] Kiczales, G., J. Rivieres, D. Bobrow. The Art of the Metaobject Protocol. MIT Press, 1991
- [19] Kiczales, G., J.M. Ashley, L. Rodriguez, A. Vahdat, D. G. Bobrow, Metaobject protocols: Why we want them and what else they can do. In Object-Oriented Programming: The CLOS Perspective, pages 101 - 118. MIT Press, Cambridge, MA, 1993
- [20] Lefkovits, H. IBM's Repository Manager/MVS, Wellesley MA:QED Information Sciences. 1991
- [21] Maes, P. Concepts and experiments in computational reflection. Conference proceedings on Object-oriented programming systems, languages and applications. pp. 147 - 155. 1987
- [22] Matula, M. NetBeans Metadata Repository (White Paper). March 2003
- [23] Object Management Group: Meta Object Facility Specification Version 1.4.
- [24] OMG. XMI - XML Metadata Interchange Specification. Version 2.0. OMG Document formal/03-05-02, May 2003

- [25] Ortner, E.. Wissensmanagement Teil 1 und 2 Informatik-Spektrum, Volume 23 Issue 2 (2000) pp 100-108
- [26] S. Danforth, I. R. Forman. Reflections on metaclass programming in SOM. Proceedings of the 9-th Conference on Object-oriented programming systems, language, and applications. pp. 440-452. 1994
- [27] SUN. JMI - Java Metadata Interface Specification Version 1.0, June 2002
- [28] Traiger, I.L., J. Gray, C. A. Galtieri, B. G. Lindsay Transactions and consistency in distributed database systems. ACM Transactions on Database Systems (TODS), Volume 7 Issue 3, September 1982
- [29] Unisys Corporation. JMI-RI Documentation. CIM Guide. Version 1.3. October 2002
- [30] Unisys Universal Repository Manager, <http://www.unisys.com/marketplace/urep/>
- [31] Wakeman, L., J. Jowett. PCTE - The Standard for Open Repositories. Prentice-Hall. May 1993
- [32] Ozsu, M., Tamer. Transaction Models and transaction management in Object-oriented database management systems, in Advances in Object-oriented Database Systems, Editors: A.Dogac, M.Tamer Ozsu, A.Bilris and T. Sellis, Series F: Computer and System Sciences, Vol. 130, 1994, Springer Verlag, New York.
- [33] Petrov, I., Stefan Jablonski, An OMG MOF based Repository System with Querying Capability - the iRM Project. Proceedings of the iiWAS Conference, September 2004
- [34] Petrov I., Stefan Jablonski, Marc Holze, Towards efficient locking of repository objects. Proceedings of IADIS Conference, October 2004.
- [35] Petrov I., Stefan Jablonski, Marc Holze, Gabor Nemes, Marcus Schneider, iRM: An OMG MOF Based Repository System with Querying Capabilities. Demo Paper. ER Conference, November 2004.

# Using Text Editing Creation Time Meta Data for Document Management

Thomas B. Hodel, Roger Hacmac, and Klaus R. Dittrich

University of Zürich, Department of Informatics,  
Winterthurerstrasse 190, CH-8057 Zürich, Switzerland  
{hodel, dittrich}@ifi.unizh.ch, hacmac@gmx.ch

**Abstract.** Word processing systems ignore the fact that the history of a text document contains crucial information for its management. In this paper, we present database-based word processing, focusing on the incorporated document management system. During the creation process of a document, meta data are being gathered. This information is generated on the level of the whole document, on sections of a document or even on individual characters and is used for advanced retrieval by so-called dynamic folders, which are superior to advanced hierarchical file systems.

## 1 Introduction

Text data (documents) are not treated as valuable data (as opposed to business data like customer, product, finance, etc.) even though a lot of companies' knowledge is stored within this structure. For a large-scale document management environment, local copies of remote data sources are often made. However, it is often difficult to monitor the sources in order to check for changes and to download data items to the copies. In many cases, text documents are stored somewhere within a confusing file structure with an inscrutable hierarchy and low security. On the other hand, data, which from an organization's point of view can be classified as crucial, is stored in databases. Here, the infrastructure and the data are highly secure, multi-user capable and available to several other tools for compiling reports, content and knowledge. Reporting and query tools can be specifically defined and applied to such data. Our idea is to make use of such a philosophy for documents. We therefore strive for the native storage of texts in a database.

Most users organize their documents by location in hierarchies onto which they map their own semantic structures. More generally speaking, hierarchies pervade document and information storage systems. The fundamental organizing principle for text documents is based on locations with the restriction of only being able to appear in one location at a time. This forces users to create strict categorizations of documents and organization. In line with these considerations, the choice of how to store and search a document has to be redefined. Users need functionality to store and locate documents without having to specify a location, and without referring to a fixed hierarchy. As a consequence, we propose that text editing creation time meta data to be automatically generated and stored (see part 2.1), which enables a

sophisticated document management system (see part 2.2) and at the same time enriches text mining functionalities.

Until now, document management and text mining solutions ignored the fact that the history of the creation process of a text document could contain crucial information. Our database-based word processing application supports not only editing but also fine-grained security, versioning, business processes, text structure, layouting, data lineage, and multi-channel publishing - all within a collaborative, real-time and multi-user environment. All of this data, along with every alteration ever made to it since its creation (especially during editing), is therefore captured [6].

In this paper, we focus on our enhanced document management system. For documents, innumerable document management systems exist [6]. According to our information no document management system has integrated automatically generated text editing creation time meta data. Realizing such a document management system involves several aspects. First of all, the word processing application has to be designed in such a way that it is able to capture all such data. This paper presents the text editing creation time meta data concept and prototype for our database-based collaborative editor. With the help of some query examples, we then demonstrate our system; the result is a completely virtualized management of text documents. The implementation of 'dynamic folders' (see part 3) provides the user with an unlimited possibility of views on documents stored in the system.

## 1.1 Problem Description

With the increasing amount of documents produced it becomes more and more important to find a way of organizing the created documents in an efficient way, so that they can easily be found when required. Currently it is easier to find and access a file created by a kid in New Zealand, than to access a file created on your colleague's desktop [9]. Crucial information contained in the large number of documents in any company or organization is at risk of being doomed to become unachievable.

The emerging market of document management and text mining systems underlines the need for tools which can manage documents in a more sophisticated way than the file system does. These systems, such as 'Documentum', 'FileNet', 'OpenText', 'Autonomy', 'SAS Text Miner', and 'Thunderstone', just to mention a few of them, have a very similar philosophy. First, they import documents, which means that the system either stores the document as it would be stored in a file system (this is the usual way), or it stores it within the database as BLOB (this is the exception). Some systems can even use both methods. Secondly, the tool analyzes the content of the document and creates a full text index. This index is normally stored within the file system.

However, current document management system solutions do not solve the problem of how to organize documents. They index the documents and can generate a response of matching documents for a specific user query using these indexes, but the underlying data organizational structure is not improved.

Even Microsoft has meanwhile recognized the necessity to revise the old-fashioned hierarchical file system with its inherent limitations. In its new Windows Version 'Longhorn', Microsoft will extend the current file system NTFS with the component 'Windows Future Storage' (Win FS) to enable access to the file system in a relational

way, as known from standard SQL databases.<sup>1</sup> The difference between Win FS and our dynamic folder concept is that we use primarily automated created metadata and not like Win FS the content of a document. Based on our knowledge, TeNDaX is the only existing database based editor and is the only system which is able to create and store all these metadata.

## 1.2 Related Work

Several papers have been written emphasizing different aspects of how to leverage document management. Some of them concentrate on possible improvements in the way that meta data could be gained from documents, while others propose new ways of organizing documents, in contrast to file systems. These are also the two main aspects which will guide us through our paper. To the best of our knowledge, no previous research paper proposes to use meta data in the way we do nor our method of organizing it. Next, some of those research papers shall be summarized.

*Placeless Documents*: The “Placeless Documents” project with its prototype “Presto” addresses the way in which documents are organized. Meta data can be assigned to every document, either manually by users or automatically by applications. The meta data of each document can then be used to create “fluid collections”: these are special folders, which automatically include or exclude documents, based on the meta data specified for the folder [2].

*KnownSpace*<sup>2</sup>: KnownSpace is an open, programmable, computational environment, suitable for arbitrary data management applications so that anyone can create anything. Small, independent programs (called simpletons) are loosely coupled with the data (called entities) and with each other. Programmers can dynamically attach arbitrary computations to arbitrary data. Simpletons parse this unstructured data, building an object-oriented database on it. The frontend may have many faces and is plugged on the KnownSpace kernel.

*Lifestreams*: A lifestream is a time-ordered stream of documents that acts as a diary of your electronic life; every document you create or receive is stored in your lifestream. The tail of your stream contains documents from the past, whereas the head of it contains more recent documents [3].

*TimeScope*: A user of TimeScope can spatially arrange information on the desktop. Any desktop item can be removed at any time, and the system supports time travel to the past (to restore desktops) and to the future (to schedule). This allows users to organize and archive electronic information without being bothered by document folders or file classification problems [10].

*Semantic File System*: A semantic file system is an information storage system that provides flexible associative access to the system’s contents by automatically extracting attributes from files with file type specific “transducers”. The automatic indexing of files and directories is called ‘semantic’ because of the use of user programmable transducers, which ‘understand’ the documents. A semantic file system

---

<sup>1</sup> <http://longhorn.msdn.microsoft.com/>

<sup>2</sup> <http://hydrogen.knownspace.org>

integrates associative access into a tree structured file system via the concept of a virtual directory. These virtual directories are interpreted as queries [4].

Looking at the process of creating meta data, none of these approaches above takes advantage of the way in which documents are created.

Many interesting approaches regarding document organization are described in the papers mentioned above. If, for example, we focus on the Placeless Documents project, fluid collections are used to organize documents. Folders do not contain documents which have been created or moved there, but rather documents whose meta data correspond with the one specified for the folder. This is a very desirable quality for the retrieval of documents, but also has a disadvantage since the content of a folder can change from one second to another. Explicit lists, which the user can specify for a folder, indicate which documents should be included or excluded in a specific folder. This somewhat alleviates the uncomfortable situation of the quickly changing content of folders.

### 1.3 Underlying Concepts

The concept of our meta data document management system requires an appropriate architectural foundation. Our concept and implementation are based on the TeNDaX [6] collaborative database based editing system.

TeNDaX is a Text Native Database eXtension. It enables the storage of text in databases in a native form so that editing text is finally represented as transactions. Under the term 'text editing' we understand the following: writing and deleting text (characters), copying & pasting text, defining text layout & structure, inserting notes, setting access rights, defining business processes, inserting tables, pictures, and so on i.e. all the actions regularly carried out by word processing users. With 'real-time transaction' we mean that editing text (e.g. writing a character/word, setting the font for a paragraph, or pasting a section of text) invokes one or several database transactions so that everything which is typed appears within the editor as soon as these objects are stored persistently. Instead of creating files and storing them in a file system, the content and all of the meta data belonging to the documents is stored in a special way in the database, which enables very fast real-time transactions for all editing tasks [7].

The database schema and the above-mentioned transactions are created in such a way that everything can be done within a multi-user environment, as is usual done database technology. As a consequence, many of the achievements (with respect to data organization and querying, recovery, integrity and security enforcement, multi-user operation, distribution management, uniform tool access, etc.) are now, by means of this approach, also available for word processing. TeNDaX creates an extension of DBMS to manage text. This addition is carried out 'cleanly' and the responding data type represents a 'first-class citizen' [1] of a DBMS (e.g. integers, character strings, etc.).

## 2 The TeNDaX Document Management Approach

We use the following terminology: whenever the term '**meta data**' is used here, it refers to the text editing creation time data. A '**text editing**' process is a logical entity that represents an editing action on a single character, on a section of a document or on a whole



document. An **'editing action'** is a task a user can do within a word processing application such as insert (write, paste), delete and change a character or characters [6], define structure, security, version, business processes, layout, insert notes, and so on [5]. Under **'creation time'**, we understand the date, time, author, roles and specific **'editing action'** information. All of these ('text editing', 'editing action' and 'creation time' data) taken together represent the stored 'meta data'. One or more 'editing actions' are combined into a **'editing action type'**. These types can be accessed and interlinked, and as a consequence, can be used for document retrieval. A **'document'** is created through an arbitrary number of 'editing actions'. The combination of all 'editing actions' assigned to a certain document in a specific order defines a current document.

## 2.1 Collecting Text Editing Creation Time Meta Data

As mentioned above, every editing action invoked by a user in the TeNDaX system is immediately transferred to the database. At the same time, more information about the current transaction is gathered.

As all information is stored in the database, one character can hold a multitude of information, which can later be used for the retrieval of documents. Meta data collected at character level are: Author, roles, date and time, copy-paste references, local and global undo / redo, security settings, version and user defined properties.

Meta data can be gained from structure, template, layout, notes, security and business process definitions. Within each section plenty of information is stored: for example the workflow section [8] contains the business process element name, its category (content, format, structure and process decision), category types (edit, verification, comment, translate, and sign) process description, date of creation, author, processors (based on users and roles), due date, time and condition, specific notes, and access rights settings. Meta data collected on the level of a document section are: author, date and time, structure affiliation, template affiliation, layout, business process affiliation, security affiliation, note affiliation, version affiliation, local and global undo / redo and user defined properties.

Last but not least, on the level of the whole document, there is meta data to be gathered during its creation and editing. Meta data we collect at the document level are: Creator, roles, date and time, document object ID, document names, structure affiliation, note affiliation, security settings, size, authors, readers, state, places within static folders and user defined properties.

All of the above-mentioned meta data is crucial information for creating content and knowledge out of word processing documents. We need these meta data for different functions within our collaborative editor, like local and global undo / redo, version, data lineage, work flow, security collaborative writing, collaborative multidimensional structuring of text and knowledge management [5], [6], [7], [8]. The next subchapters show how this meta data can be used for document retrieval and how the TeNDaX document management system works.

## 2.2 The Usage of Text Editing Creation Time Meta Data for Document Retrieval

Using the meta data gained, the following example queries can be asked in the TeNDaX document management system. The concrete form of querying and presentation of results is discussed in the following chapters.

- Show all documents in workflows which have pending tasks for me, or which I have written and which have been rejected in a workflow by another user.
- Show all documents of which more than 50% was written by user “Dittrich” and which haven’t been modified since 1.1.2001.
- Show all documents to which I have write rights and which have been read by more than 100 users with the role “Employee”.
- Show all documents which have character security restrictions for the role “Employee”.
- Show all documents edited by team “A” and read by my boss last week?
- Show all documents which have been written by user “Hodel” or “Hacmac”, with creation date after 1.1.2000 and size more than 1000 characters.
- Show all the documents accessible by the role “Employee”, which have the document name “\*proj\*”.
- Show all documents written by myself, with similar sentences and phrases to the document “Project TeNDaX”.
- Show the document with ID “1230” and all documents with copy-paste references to this document, created by user “Hodel”.
- Show all documents which are somewhere in a directory called “TeNDaX” and which are marked by the user defined property “to be done”.
- Which documents were read or printed out by our project manager, and when?
- Who wrote this paragraph originally?
- Which parts have been copied and pasted, and from which source?
- As a final example, we can look at the following situation which could occur in a company: four documents containing relevant knowledge for an upcoming project are found. Based on the information about which part was written by which author, and which part was copied from another document, the system can pinpoint suitable employees and teams to discuss the new project.

## 2.3 Document Organization: Static and Dynamic Folders

The question arises as to whether this meta data can be used to create an alternative storage system for documents. As discussed in part 1.2, several papers have been written on how to improve document management. In any case, it is not an easy task to change users’ familiarity to the well known hierarchical file system.

This is also the main reason why we do not completely disregard the classical file system, but rather enhance it. Folders which correspond to the classical hierarchical file system, will be called “static folders”. Folders where the documents are organized according to meta data, will be called “dynamic folders”. As all information is stored in the database, the file system, too, is based on the database.

### 2.3.1 Static Folders

Static folders are folders as common. Users can create them, modify their names, delete, copy and move them and assign access rights if they are authorized to do so. Their main function is to store documents users create with the TeNDaX word processor.

Static folders are organized as follows:

- There is one private folder for each user which represents the user’s private area. Under no circumstance can other users access this folder. The user may carry out any actions he wants to, e.g. create, modify, copy, move or delete folders and documents. A private folder is quite similar to stored files on a ‘local disc’.
- For each role created, a public static folder is set up automatically by the system. A user is only able to see a public static folder if he was assigned to the corresponding role. The rights of the user in this case depend on the rights given to him by the administrator. Example: a new project is started and a new role (i.e. user group) is generated for this project. Next, all the people involved are assigned to this role, so that shared access to this public static folder is granted. These folders are quite similar to sets of files stored on a ‘file-server’ or within a document management system.

### 2.3.2 Dynamic Folders

The place where the meta data can be used for document retrieval, are the dynamic folders. The dynamic folders build up sub-trees, which are guided by the meta data selected by the user.

Thus, the first step in using a dynamic folder is the definition of how it should be built. For each level of a dynamic folder, exactly one meta data item is used to. The following example illustrates the steps which have to be taken in order to define a dynamic folder, and the meta data which should be used.

**Table 1.** Defining dynamic folders (example)

Level	Meta data	Restrictions	Granularity
1	<i>Creator</i>	Only show documents which have been created by the users “Hodel” or “Dittrich” or “Hacmac”	One folder per creator
2	<i>Business process affiliation</i>	Only show documents with closed tasks for the user group “Manager”	One folder per task status
3	<i>Business process affiliation</i>	Only show documents with tasks completed by the user “Meier”	One folder
4	<i>Authors</i>	Only show documents where at least 40% was written by user ‘Hodel’	Each 20% one folder
5	<i>Structure affiliation</i>	Only show documents which have been assigned the template “LNI” or “LYNX”	One folder per template
6	<i>Copy-paste references</i>	Only show documents which have no copy-paste references to a document with the name “DKE TeNDaX”.	One folder

As a first step, the meta data which will be used for the dynamic folder must be chosen. As we see in Figure 1, the sequence of the meta data influences the structure of the folder. Furthermore, for each meta data used, restrictions and granularity must be defined by the user; if no restrictions are defined, all accessible documents are listed. The granularity therefore influences the number of sub-folders which will be created for the partitioning of the documents. Figure 1 visualizes the dynamic folder defined in Table 1. We named this dynamic folder ‘DF\_Paper’.

As the user enters the tree structure of the dynamic folder, he can navigate through the branches to arrive at the documents he is looking for. The directory names indicate which meta data determines the content of the sub-folder in question. At each level, the documents, which have so far been found to match the meta data, can be inspected. This is symbolized in figure 1 by the little document icons below each folder. For the folders “Creator\_Hodel” and “Creator\_Hacmac”, the same structure as that which is shown for the folder “Creator\_Dittrich” would be constructed.

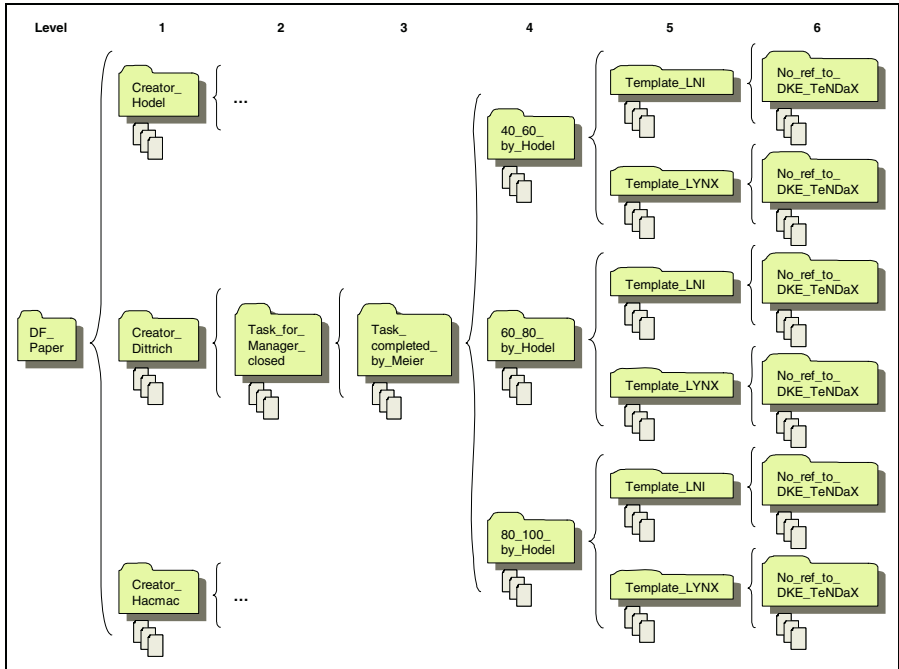


Fig. 1. Representation of a dynamic folder

Ad hoc changes of granularity and restrictions are possible in order to maximize search comfort for the user. It is possible to predefine dynamic folders for frequent use, as well as to create and modify dynamic folders on an ad hoc basis. Furthermore, the content of such dynamic folders can change from one second to another, depending on the changes made by other users at that moment.

### 3 Prototype

#### 3.1 Collecting and Storing Creation Time Meta Data

Some of the meta data used for document management is deliberately gathered for this purpose only. On the other hand, some of it originates from other functionalities of the word processor. These functionalities are optimized for best performance. This

is the reason why the meta data for document management is not stored in a central area of the database, but is rather widely distributed. Here, the aspects of operational functioning and data warehousing are combined in the sense of a hybrid approach. Figure 2 shows which attributes are located in which database classes. The figure only shows the attributes relevant for document management; other attributes are omitted. The associations between the classes are not symbolized by lines, but instead with attributes in the classes of the type of the associated class.

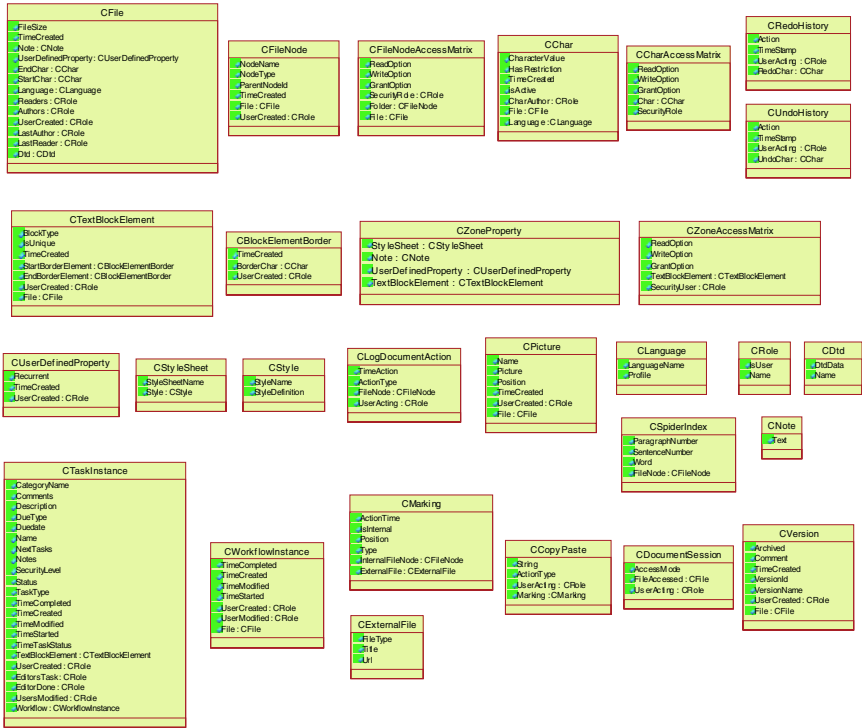


Fig. 2. Database schema showing only attributes relevant for document management

In part 2 we outline which meta data is held. The meaning of the attributes in the classes shown in the database diagram should be explanatory.

Obviously, the different attributes cannot all be of the same type nor can they have the same scopes. Thus for each attribute a separate method handles the scope and granularity. The next chapter shows how these meta data are handled finally in order to create the dynamic folders.

### 3.2 Generating the Dynamic Folders

As described above, each level of a dynamic folder represents a partitioning of the documents regarding some meta data, each level partitioning the previous one. Thus,

the first step a user must take to be able to use the dynamic folders, is to define the desired dynamic folders. This definition includes which meta data is decisive for which level of a dynamic folder, which conditions regarding this meta data the documents must accomplish, and with which granularity the sub-folders should be created. Remember: only one meta data can be chosen for each level of a dynamic folder. Figure 3 describes the process of opening the first level of a dynamic folder.

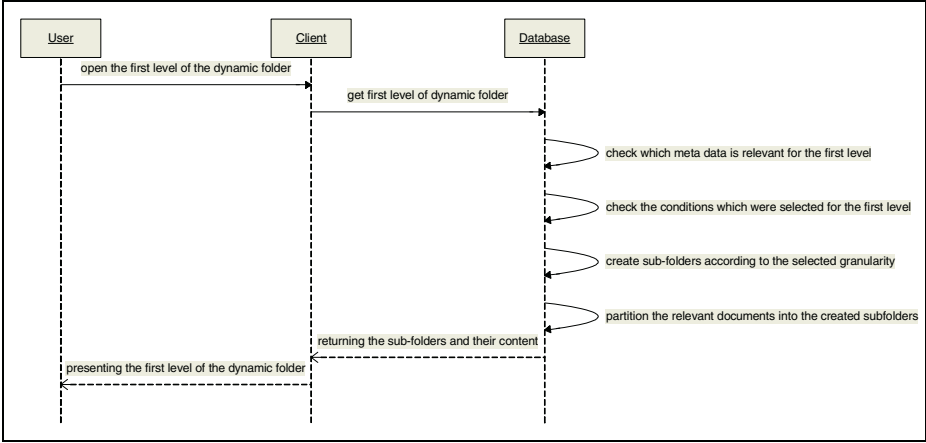


Fig. 3. Sequence diagram ‘opening the first level of a dynamic folder’

When the first level of the dynamic folder is presented to the user, he can then choose to open the next level. The processing of this action is equal to the one shown in figure 3, with the difference that the documents which have now to be considered are only those from the sub-folder the user came from, and not from the whole document base.

The following lines describe how the meta data is used to create database queries, i.e. how the relevant documents are selected and filed in the sub-folders of the dynamic folder.

In the following description, the objects and functions used for creating the dynamic folders are portrayed. (The elementary functions are assumed to exist.)

The symbol  $d$  stands for the object “document”.

$$d = \text{document}$$

The function *collect* collects all the documents available to and accessible by the acting user *user*. This includes checking the security restrictions, so that documents for which the user has no access rights are not included into the document base *db* for that specific user. The collection of documents *db* is therefore a set of documents *d*. The function *collect* is only called at the first level of a dynamic folder. The method used for the higher levels is described later on.

$$db = \{d_1, d_2, \dots, d_n\} = \text{collect}(\text{user}) ; n = \text{number of documents found}$$

The symbol  $m$  stands for the meta data and symbolizes one of the selectable meta data, as described in part 2.1. For reasons of space, not all the meta data are listed here, but all of these are valid, regardless if they concern the character, section or document level.

$$m = \exists \{Author, Roles, \dots, User\ defined\ properties\}$$

The function *validate* validates if the documents in the document base  $db$  accomplish the conditions which were specified for the meta data, which is decisive for the current level of the dynamic folder. The scope of validity is defined in the object  $sc$  and must correspond to the type of meta data it applies to. The result of the function *validate* is a reduced document base  $db'$ .

$$db' = validate(db, m, sc)$$

As the relevant documents are now isolated in  $db'$ , we need a function which builds up the sub-folders in the dynamic folder, as required by the granularity which the user has chosen. The function *partition* takes as its parameters the document base  $db'$  valid for the current user, the meta data  $m$  which is decisive for the current level, and the granularity  $g$  which decides upon the amount of sub-folders  $sf$  to be created. Also the granularity  $g$  must correspond to the type of meta data it applies to. The result of this function is a set of sets, i.e. documents filed correctly into the sub-folders.

$$\{sf_1, sf_2, \dots, sf_x\} = partition(db', m, g); x = \text{number of sub-folders created}$$

$$sf = \{d_1, d_2, d_y\}; y = \text{number of documents filed into this sub-folder}$$

The result of the function *partition* is passed to the client. As the user dives into one of the created sub-folders, the same procedure of validating and partitioning takes place. The input for the functions *validates* and *partition* is then:

$$db = sf_z; z = \text{sub-folder selected by the user, } z = \exists \{1, 2, \dots, x\}$$

$m =$  meta data relevant for the next level of this dynamic folder, as defined by the user

$$sc = \text{scope for the meta data } m, \text{ as defined by the user}$$

$$g = \text{granularity for the meta data } m, \text{ as defined by the user}$$

If we consider the example from part 2.3.2, the function *validate* would be called with the following parameters for the first level of the dynamic folder:

$$db' = validate(db, "Creator", "Hodel or Dittrich or Hacmac")$$

The corresponding SQL code is:

```
SELECT CFileNode.ID As fnID, CFile.UserCreated As Creator
FROM CFileNode INNER JOIN CFile ON (CFileNode.File =
CFile.ID) INNER JOIN CRole ON (CFile.UserCreated =
CRole.ID)
```

```
WHERE (CFileNode.IsDynamic = 0) AND ((CRole.Name =
"Hodel") OR (CRole.Name = "Dittrich") OR (CRole.Name =
```

```
"Hacmac" ) )
    IsDynamic = 0: static documents
```

For this purpose, the function *validate* uses the attribute *UserCreated* from the class *CFile* to evaluate the necessary meta data. The document base *db* is generated by the function *collect* with the help of the class *CFileNodeAccessMatrix*, which is responsible for all security issues on the document level. For the second level, *validate* is called as follows. *s* represents the sub-folder selected by the user.

```
db' = validate(sf, "Business Process Affiliation", "Documents with closed tasks to
the user group 'Manager'")
```

The corresponding SQL code is:

```
SELECT CFileNode.ID As fnID
FROM CFileNode INNER JOIN CFile ON (CFileNode.File =
CFile.ID) INNER JOIN CWorkflowInstance ON (CFile.ID =
CWorkflowInstance.File) INNER JOIN CTaskInstance ON
(CWorkflowInstance.ID = CTaskInstance.Workflow) INNER
JOIN CRole ON (CTaskInstance.EditorsTask = CRole.ID)
WHERE (CFileNode.NodeType = 2) AND (CFileNode.IsDynamic
= 1) AND (CRole.Name = "Manager") AND
(CTaskInstance.Status = 1) AND (CFileNode.ParentNodeId
= [Node ID of selected dynamic sub-folder])
CTaskInstance.Status = 1: task closed
CFileNode.IsDynamic = 1: dynamic documents
```

In this case, the meta data used originates from the attribute *EditorsTask*, in the class *CTaskInstance*.

The steps discussed above would also apply to the next levels of the dynamic folder. For each meta data, the functions *validate* and *partition* provide specialized methods which use the necessary classes.

## 4 Conclusion

The dynamic folder concept is a high-level data model. People use a computer to communicate and to store and organize their personal data. Unfortunately, a computer does a relatively poor job of allowing users to organize the information so that it can be found later. When a user forgot exactly where he puts a file, it can take quite a while to find it again. In the worst case, the entire content of each disk has to be searched.

One reason why to find information on a computer is difficult, is because of the limited ability for the user to organize data. The hierarchical folder structure does not work well when a categorization of data in numerous ways is wanted. Therefore, the first problem is that lots of files have to be stored and no good way to categorize them is available. Another problem is that the same stuff is stored in multiple places in multiple formats. There are a number of problems with current approaches to data



storage, like: Multiple applications cannot share common data, the same information lives in multiple locations, separate copies of data become unsynchronized, and there are no notifications of data change.

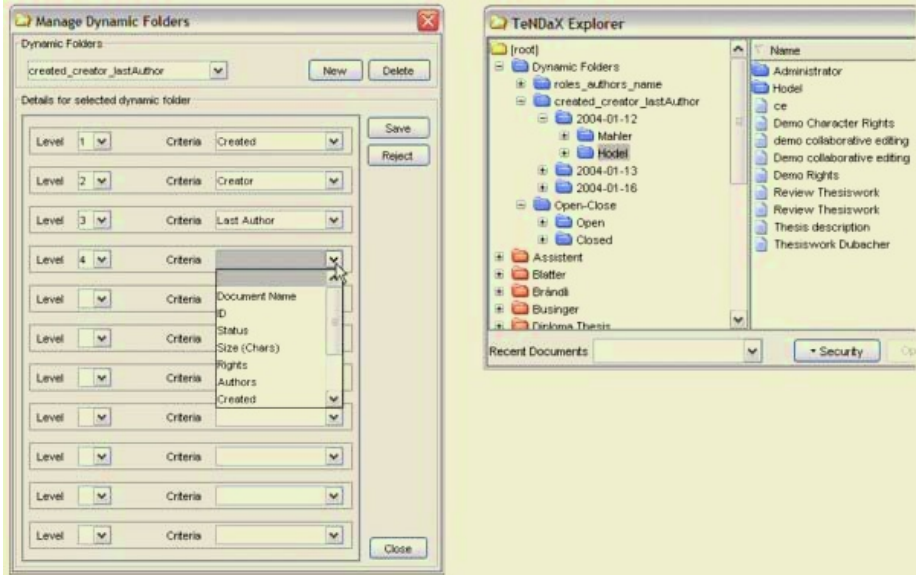


Fig. 4. TeNDaX screenshots

Dynamic folders improve text documents in three ways. First, they store automatically all thinkable meta data and relate one item of information to another. Second, it provides a common storage format for information collected. Third, it promotes data sharing of common information across multiple applications. Dynamic folder is an active storage platform for organizing, searching for, and sharing all kinds of information. This system defines a rich data model that allows using and defining data types that the storage platform can use. All these features together allow four ways to organize documents with dynamic folders: Hierarchical folder-based organization, item property-based organization, relationship-based organization, category-based organization.

TeNDaX reached the status of a quite stable prototype. The system is used in several pilot projects and within some courses at different universities. The described dynamic folders are implemented and running quite well (see Figure 4). Performance is similar to known file-explorer and therefore not an interesting issue. Integration into the Windows-Explorer was programmed too. Further information and a demonstration video clip can be found on the TeNDaX<sup>3</sup> website.

In this paper we have proposed a dynamic document management system environment that represents all documents in a database system, working with the

<sup>3</sup> <http://www.tendax.net/>

underlying TeNDaX architecture. This architecture enables different views of documents in a structured, reliable and real-time co-operation environment.

## References

1. S. Abiteboul, R. Agrawal, P. Bernstein, M. Carey, S. Ceri, B. Croft, D. DeWitt, M. Franklin, H. G. Molina, D. Gawlick, J. Gray, L. Haas, A. Halevy, J. Hellerstein, Y. Ioannidis, M. Kersten, M. Pazzani, M. Lesk, D. Maier, J. Naughton, H. Schek, T. Sellis, A. Silberschatz, M. Stonebraker, R. Snodgrass, J. Ullman, G. Weikum, Widom, and J. Stan Zdonik, "The Lowell Database Research Self Assessment," Massachusetts 2003.
2. P. Dourish, W. K. Edwards, J. Howell, A. LaMarca, J. Lamping, K. Petersen, M. Salisbury, D. Terry, and J. Thornton, "A programming model for active documents," proceedings of the 13th annual ACM symposium on user interface software and technology, New York, USA, 2000.
3. E. T. Freeman, *The Lifestreams Software Architecture*: Yale University Department of Computer Science, 1997.
4. D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole, "Semantic File Systems," proceedings of 13th ACM Symposium on Operating Systems Principles, 1991.
5. T. B. Hodel, D. Businger, and K. R. Dittrich, "Supporting Collaborative Layouting in Word Processing," proceedings of IEEE International Conference on Cooperative Information Systems (CoopIS), Larnaca (Cyprus), 2004.
6. T. B. Hodel and K. R. Dittrich, "Concept and prototype of a collaborative business process environment for document processing," *Data & Knowledge Engineering*, vol. Special Issue: Collaborative Business Process Technologies, 2004.
7. T. B. Hodel, M. Dubacher, and K. R. Dittrich, "Using Database Management Systems for Collaborative Text Editing," *ACM European Conference of Computer-supported Cooperative Work (ECSCW CEW 2003)*, Helsinki (Finland), 2003.
8. T. B. Hodel, H. Gall, and K. R. Dittrich, "Dynamic Collaborative Business Processes within Documents," proceedings of ACM Special Interest Group Conference on Design of Communication (SIGDOC) 2004, Memphis (USA), 2004.
9. IBM On Demand Workplace, "How electronics companies can become more resilient and adaptable in a chaotic world," IBM Business Consulting Services, 2003. [http://www-1.ibm.com/services/us/igs/pdf/ibm\\_ondemand\\_workplace\\_electronics\\_18feb04.pdf](http://www-1.ibm.com/services/us/igs/pdf/ibm_ondemand_workplace_electronics_18feb04.pdf)
10. J. Rekimoto, "TimeScape: A Time Machine for the Desktop Environment," proceedings of CHI'99 late-breaking results, 1999.

# An Object-Relational Approach to the Representation of Multi-granular Spatio-Temporal Data

Elisa Bertino<sup>1</sup>, Dolores Cuadra<sup>2</sup>, and Paloma Martínez<sup>2</sup>

<sup>1</sup>CERIAS and CS Department, Purdue University, Recitation Building,  
656 Oval Drive, West Lafayette, IN 47907-2086, USA.

bertino@cerias.purdue.edu

<sup>2</sup>CS Department, Carlos III University, Avd. Universidad 30, 28911 Leganés, Spain

Fax: 34-91-6249430

dcuadra@inf.uc3m.es, pmf@inf.uc3m.es

**Abstract.** The notion of spatio-temporal multi-granularity is fundamental when modeling objects in GIS applications in that it supports the representation of the temporal evolutions of these objects. Concepts and issues in multi-granular spatio-temporal representations have been widely investigated by the research community. However, despite the large number of theoretical investigations, no comprehensive approaches, have been proposed dealing with the representation of multi-granular spatio-temporal objects in commercially available DBMSs. The goal of the work that we report in this paper is to address this gap. To achieve it, the paper first introduces an object-relational model based on OpenGis specifications described in SQL3. Several extensions are developed in order to improve the semantics and behavior for spatio-temporal data types introducing an approach to represent the temporal dimension in this model and the multi-representation of spatio-temporal granularities.

## 1 Introduction

Since 1970 when Codd [5] proposed the relational model, database system technology has introduced several important changes in the way data are stored and managed. The relational data model has had a great impact on commercial products, mainly because of the development of the SQL language, which included several additional features with respect to those specified by the theoretical definition of the relational model. Since its initial definition, SQL has been widely extended [9]. A relevant set of extensions has dealt with the introduction of object modeling capabilities [21], resulting in the notion of object-relational data model. Such a model combines the simplicity and the power of SQL the ability of describing new data types with their associated operations, typical of the object-oriented approach. The object-relational data model is thus a powerful model combining the best aspects of two different approaches.

The SQL3 standard [10] is the reference language for the object-relational model. It has been defined by extending the previous SQL92 standard with the ability to modify, retrieve and define the data types needed to represent a large variety of application domains. Examples of those extensions include: XML, MULTISSET (like the ARRAY data type, without implicit order), BIGINT and others.

An important class of applications is represented by spatial and GIS applications. These applications are relevant in a number of different domains, such as transportation, urban planning, homeland security, and environmental protection. Spatial data management is often combined with temporal data management, because in many cases one needs to record the temporal evolution of spatially-related entities. The resulting data models are thus termed spatio-temporal data models. One of the most crucial issues when dealing with spatio-temporal databases is the management of the information concerning moving objects in a spatial context. Such an issue represents an important requirement in several application domains, like air traffic control and habitat control of endangered species and so on. So far, the GIS systems have handled the spatial and the non-spatial data separately, which increases the complexity of maintaining data integrity. The use of an object-relational database system to manage this kind of data represents a good alternative, in particular because such a system is able to homogeneously and efficiently manage user-defined information, and to improve integrity for data of different nature.

Most of the spatial-temporal data approaches proposed so far do not exploit the powerful modeling and management features that are provided by recent versions of commercially available DBMSs. Another main limitation of current approaches is that they do not support multiple granularities in the representation of spatio-temporal data. Multiple granularities, defined as a set of measure units for space and time, are crucial in facilitating the management of information for applications such as air traffic control, meteorological forecast and so forth [3], [17]. The goal of the work reported in this paper is to address such limitations by developing an object-relational approach to the management of spatio-temporal data supporting multiple granularities for both space and time.

In particular, in the paper we propose a temporal extension expressed in the SQL3 standard, specifically tailored to the OpenGis specifications [29]. Our start point is the object-relational model and meta-model that verifies the OpenGis specifications from which we develop an extension of the spatial and non-spatial data types defining a new data type composed the two parts; the first part is the object value and the second the valid time for this value. This extension provides the support required to model multiple spatio-temporal granularities and tools to operate on objects with different granularities in order to address their integration and inter-operability.

The remainder of this paper is organized as follows. Next section gives an overview of related work dealing with the spatial and temporal representation in the object-relational model. Section 3 introduces the notions of spatial and temporal granularity and their basic properties. Section 4 describes the OpenGis specifications in SQL3 [29] and presents our extension to supporting multiple spatial granularities. Section 5 then extends the model and meta-model developed in Section 4 by introducing time as an extension of the SQL3 data types. The last section concludes the paper and outlines future work.

## 2 Related Works

Initial approaches, aimed at supporting advanced data management applications, have recognized that spatial information [15], [24] and temporal information [25], [22],

[23], [19] are both crucial. However, those early approaches have dealt with space and time representations separately. Most recent proposals have proposed integrated approaches able to model both spatial and temporal aspects of data objects.

A first relevant approach has been developed by Guting et al. [16]. The approach supports the change of the position or extension for geometry through the use of abstract data type definition capabilities. It develops a set of constructors and query operators in an abstract model thus giving a compact and uniform vision for every data type. In the proposed approach [16], some data types, namely Integer, Boolean, Spatial, can be transformed in a temporal data type. Such an abstract model has been then transformed in a discrete model [14], closer to the implementation but more restricted with respect to the abstract model. The discrete model represents the object values that have a temporal dimension through the use of snapshots. This method is thus referred to as *sliced representation*. As part of their work, Guting et al. [14] have shown how the sliced representation can be mapped onto relational data structures such as records and arrays. However, they do not have specifically addressed how to map their abstract model onto the SQL3 standard.

Chen and Zaniolo [6], based on the spatial-temporal representation model proposed by Worboys [26], propose a number of SQL3 extensions aiming at supporting advanced queries. Unlike the previous proposals, they adopt a point-based approach to model the time dimension and queries are expressed through user-defined aggregate functions. Such an approach to handle time for spatio-temporal databases represented in a relational framework, it is very simple and minimizes SQL3 extensions in comparison with the complex functions one has to apply when using based-intervals approaches [13], [19]. Such an approach has then been architected by solutions supporting efficient storage [7]. Finally, a more recent paper [8] by the same authors describes how to implement these functions in ATLAS [28] in order to model specific application domains. This approach keeps the SQL philosophy; therefore it is easy to use for developers and users. A major drawback of this approach is that it is not able to support multiple temporal attributes with different granularities in the same relation.

Another approach very close to implementation has been proposed by Lee [18]. The spatial object history is modeled through the creation of relations. The developed framework is based on the creation of some special purpose relations, called *dimensions*, storing the current values of the object geometry. Besides those dimension relations, the approach requires the introduction of two additional relations keeping historical information. The approach by Lee also includes a set of macro operators that are used to execute queries. The macro operators are applied to spatial data, temporal data, and spatio-temporal data. Also, an aggregate operator is defined to manage the history of a spatio-temporal data. The main drawback of this approach is that it requires executing join operations among the dimension tables in order to produce complete spatial information. The use and maintenance of data under such an approach is thus quite complicated.

An extensive analysis of the most important approaches to represent spatio-temporal in the object-relational model has been carried out by Erwig et al. [12]. This paper presents an extensive comparison between two approaches to represent the temporal dimension in a spatio-object-relational model. The first approach is based on extending each relation storing spatial data objects, with an additional column

representing temporal information. The column essentially records the temporal validity of row values. The second approach exploits the expressive power of the object-relational model. It is based on the definition of an abstract data type (ADT) joining space and time on the same column. The second approach better represents the semantics of the original data and allows one to create relations containing time information at column level.

Compared to previous work, the main contributions of our work are summarized as follows. First, the spatial representation we adopt is based in the proposal of OpenGis specifications described through SQL3 [29]. The above approaches, by contrast, adopt much simpler spatial data models not supporting OpenGis. Second, we provide support for multiple spatial granularities. Third, we use an abstract data type approach to represent and manage the temporal dimension; thus our approach is more flexible than other approaches and supports a finer degree of control over temporal behavior. Our approach facilitates the formulation of queries in SQL and provides a representation which is more intuitive and easy to use. Finally, we propose solutions to maintain multiple granularities, in both space and time, and provide conversion functions to map data representations among different granularities. None of the above described approaches provides such functions.

### 3 Preliminaries: Multi-granular Spatio-Temporal Representation

In this section, we introduce the relevant notions underlying our approach to multiple granularities in both time and space aspects. From an informal view point, we can think of a temporal or spatial granularity as a discrete partition of time or space. Different partitions may be defined, thus resulting in multiple granularities.

The temporal granularity has been defined as the partitioning of a temporal domain in groups of elements ordered through an index set, where each group is perceived as an indivisible unit (a *granule*) [2]. The granularities set is denoted by  $G_T$  and its elements are related by the relationship *finer\_than*. A granularity  $S$  is *finer\_than*  $R$  if for each index  $i$  a index  $j \in R$  exists such that  $S(i) \subseteq R(j)$ , where  $S(i)$  denotes the granule belonging to the  $i$  index position. This relation is denoted by  $S \leq_T R$ . Semantically, when we define a granularity for an object we specify the time instants at which the object's values are relevant to the specific application domain. Days, months and years are several kinds of temporal granularity; days are finer than months and months are finer than years. Operations and comparisons between objects at different temporal granularity require the use of *conversion functions*. These functions change the temporal properties of an object from a finer granularity to a coarser granularity and we can compose a macro function with the function composition. Conversion functions can be like the ones shown in the Table 1, or aggregation functions available in SQL [4] or created according to the application's semantics.

The spatial granularity is defined as the unit of measure in a spatial reference system. It thus represents the unit according to which spatial properties, like the area, are measured. The set of the spatial granularity is denoted by  $G_S$ . The elements of  $G_S$  would be meters, kilometers, grades and others. Each one of those elements must be defined with respect to a spatial reference system. In  $G_S$ , the relation *finer\_than* is similar to the temporal granularity given before and it is denoted by  $M \leq_S N$ .

**Table 1.** Temporal and spatial conversion functions

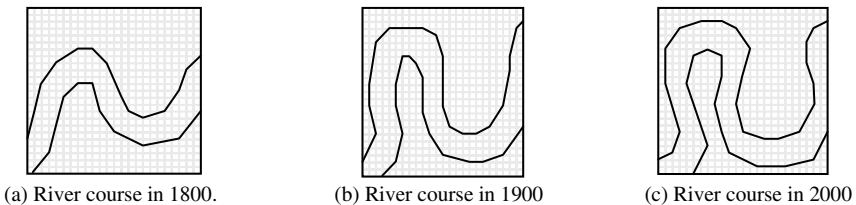
<b>Proj (index)</b>	It returns, for each granule in the coarser granularity, the value corresponding to the granule of position index at the finer granularity	<b>Contract functions</b>	
<b>First, Last</b>	First and last index in the Proj (index) function	l_contr	It contracts an open line, endpoints included, to a point
<b>Main</b>	It returns, for each granule in the coarser granularity, the value which appears most frequently in the included granules at the finer granularity	r_contr	It contracts a simple connect region and its bounding to a point
<b>All</b>	It returns, for each granule in the coarser granularity, the value which always appears in the included granules at the finer granularity if this value exists, the null value otherwise	r_thinning	It reduces a region and its bounding lines to a line
		<b>Merge functions</b>	
		l_merge	It merges two lines sharing an endpoint into to single line
		r_merge	It merges two regions sharing a boundary line into a single region
		<b>Absorption operations</b>	
		P_abs	It eliminates (abstracts) an isolated point inside a region
		l_abs	It eliminates a line inside a region

Possible conversion functions are shown in Table 1; we refer the reader to [1] for additional details. The application of these conversion functions guarantees the topology consistency.

Suppose we were interested in studying the evolution of a river’s course. According to this specification, we can define a time unit as century thus obtaining the representation shown in the Fig. 1. Semantically speaking, the temporal and spatial granularity integration provides, on the one hand, the timestamp at which a spatial object is observed and, on the other hand, establishes the spatial measure within a reference system.

Our approach represents spatial-temporal multiple granularities considering the spatial representation of an object with respect to one temporal granule. Therefore, we must specify the observation unit for both space and time. These objects are called *moving objects* [16]. The Fig. 1 shows the time variation in the river course with centuries as temporal granularity.

The temporal and spatial multi-representation is built when executing user queries by using the conversion functions defined for each application depending on the domain. In the next section, we show how to specify the conversion functions



**Fig. 1.** River course evolution

composition and their composition order what we denote as *composition sequence*. The composition of functions is strongly domain-dependent and it is very important to store it with the scheme definition to provide more semantics.

## 4 Spatial Framework: Modeling OpenGis Specifications with SQL3

In this section, we describe the OpenGis specifications in SQL3 considering the abstract data types to define the Geometry type besides the meta-scheme definition necessary to manage the spatial data in an object-relational database.

Our approach begins with the framework developed in [29] and provides an extension through the new meta-information and SQL3 functions to support the spatial granularity. The proposal is focused on this spatial framework for two reasons; first, the representation agrees with the OpenGis standard and, second, it is specified by another standard, the SQL3 language, which provides an easy implementation in any object-relational DBMS.

### 4.1 Spatial Data Description

We describe OpenGis from two different viewpoints. The first describes the geometry data types, their properties and functions related to the spatial relationships among geometric objects. The second describes the meta-scheme which gives support to the geometry data type. The meta-scheme is similar to a data dictionary. It is composed of relations describing the spatial domain. The meta-schema in [29] describes the spatial domain as geometric objects in  $\mathcal{R}^2$  space.

The relations belonging to the meta-schema cannot be modified by the user. The relation definition is fixed in the meta-schema and modifications to it are achieved through the use of views. We consider a view as a subset of meta-information that can be modified by the user.

In the spatial meta-scheme presented in [29], framework of our proposal, when we define a geometric object in an object-relational scheme, the meta-scheme stores the information concerning its dimension, its geometry type (if it is a point, a line, a polygon and so on) and the spatial reference system over it is defined. This information allows, for example, one to check whether the relationships among geometric objects can be determined. The spatial relationship functions can just apply to objects with the same granularity and spatial reference.

We introduce a notation to reference this meta-information. We denote by SOR the spatio-object-relational model and by SOR-M the meta-scheme over SOR defined in [29].

**Definition 1:** The SOR-M =  $\{RM_1, \dots, RM_n\}$  is a set of relations that describe the spatial domain. We define  $V_s$  as a set of views,  $V_s = \{V_{s_1}, \dots, V_{s_i}\}$  such that  $V_{s_j} = \text{Ops}(RM_k, \dots, RM_j)$  where  $\text{Ops}$  is a macro-operator composed of relational algebra operators.  $\square$

The meta-information held in  $V_s$  can be modified by user. We reference to SOR-M as the set of views ( $V_s$ ).



**Definition 2:** The Geometry data type ( $G$ ) is defined as an abstract data type whose features<sup>1</sup> are described in two important views, *Geometry\_Columns* and *Spatial\_Ref\_Sys*, such that  $Geometry\_Columns, Spatial\_Ref\_Sys \in V_s$ .  $\square$

The geometry type in *SOR* model is represented using a discrete model because *SOR* is a model close to the implementation. The spatial representation is stored as a set of vertexes that are interpreted through the linear interpolation between them.

According to Fig. 2, the *Geometry* type has a hierarchical structure that is specialized into Point, Curve, Surface and Geometry Collection subtypes. Fig. 2 is based on Geometry Model specified in the OpenGIS Abstract Specification restricted to 0, 1, and 2 dimensions for the collection types named Multipoint, MultiLineString and MultiPolygon. Therefore, Geometry, Curve, Surface, Multicurve and Multisurface are introduced as abstract data types. The subtypes of Geometry are restricted to 0, 1, and 2 dimensions in a coordinate space ( $\mathcal{R}^2$ ).

The geometry type includes basic functions that retrieve information about spatial properties and methods for testing the spatial relationships and analyzing geometric objects; besides for each type specific functions are defined. In this paper, we do not show these functions and methods because they are specified in [29]. We focus our paper on how we can define a geometry object in a relation and how to represent its spatial properties in *SOR-M*.

**Definition 3:** Let  $R (A_1:D_1, \dots, A_n:D_n)$  be a relation where  $A_i$  is a column defined over a domain  $D_i$ .  $R$  is a *feature relation* if  $\exists A_i (i \in \{1, \dots, n\})$  such that  $A_i \in G$ .  $A_i$  is called *geometry or spatial column*.  $\square$

When we define a *feature relation*, the spatial attribute definition shows just the geometry type. In the *SOR* model, every geometry column stores its features in the *Geometry\_Columns* and it is referred to *Spatial Reference System (SRS)*. A *Spatial Reference System* (also called *Coordinate System*) is a way to assign coordinates to a location and to establish relationships between sets of such coordinates. It enables the interpretation of a set of coordinates as a representation of a position in a real world space. Any spatial data in *SOR* has a coordinate system associated with it through the *Spatial\_Ref\_Sys* view.

To define the geometry column features we introduce procedures to insert these features in *SOR-M* in order to make the update operations easy for the users. The *Spatial\_Ref\_Sys* view has defined SRS by default but the user can define other SRSs. For this reason, we create two SQL 3 procedures.

**Definition 4:** Let  $R (A_1:D_1, \dots, A_n:D_n)$  be a feature relation. We define geometry features  $\forall A_i \in G$  through the following procedures:

- *SP* ( $R, A_i, Dimension\_Number, DRS\_ID$ ) is a SQL3 procedure which inserts the dimension number and the identifier DRS ( $DRS\_ID$ ) in the *Geometry\_Columns* view when  $A_i$  is defined in a SRS by default (DRS)
- *SR* ( $SRID, SRText$ ) is a SQL3 procedure to create a new spatial reference system. It inserts the identifier reference system and the specification textual in *Spatial\_Ref\_Sys*. The *SRText* column describes a standard specification defined in [29].  $\square$

---

<sup>1</sup> We use the term feature to indicate an attribute of an abstract data type.

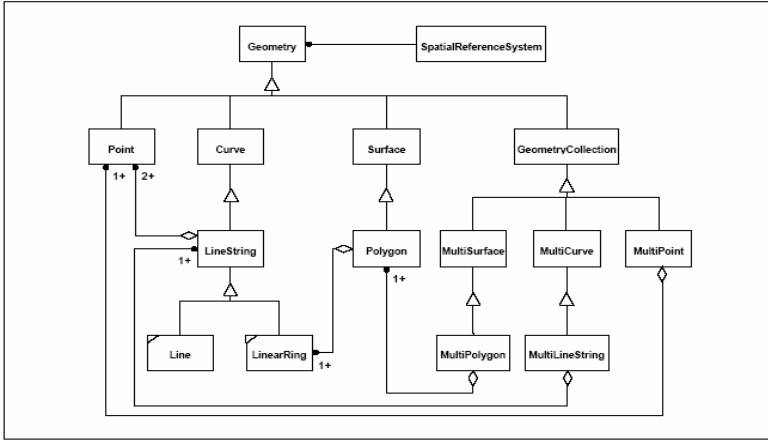


Fig. 2. Geometry description in ODMG [21]

Example 1: Let *City* be a relation defined with two attributes, *Name* of Varchar type and *Extension* of Polygon type. We define the *City* feature relation as:

```
City(Name:varchar(60), Extension:Polygon);
```

The *Extension* column is defined as a Polygon subtype. If we want to define the *Extension* column in *City* as a polygon with two dimensions at meters in a spatial reference system by default (*DRS\_ID* = 14), we must specify them with the *SP* procedure to insert these features in the *Geometry\_Columns* view:

```
SP('City', 'Extension', 2, 14);
```

The *SP* signature indicates with the number 2 that *Extension* is a two-dimensional geometry object having 14 as identifier for the spatial reference system (*DRS\_ID*). This identifier references a row of *Spatial\_Ref\_Sys* view. □

### 4.2 Spatial Granularity Description and Proposed Extensions by *SOR* Model

The spatial granularity is a measure unit in relation to a *Spatial Reference System* (*SRS*). In the previous spatial framework, the spatial granularity is referred to the *SRS* chose. We can distinguish between a spatial granularity referred to a default reference system (*DRS*) and a new spatial reference. The *DRS* supported in the OpenGis specifications appears in [29] and the granularity is always defined according to the default unit. The definition of a new reference system involves the insertion of a new row in *Spatial\_Ref\_Sys* view. In order to provide an easy interaction with the *SOR-M* to define a spatial granularity different with respect to the *Unit* parameter defined over *DRS* we create a new granularity procedure.

**Definition 5:** Let *SOR* be a model, we define *SG* procedure such that *SG*(*Table\_Name*, *Geometry\_Name*, *DRS\_ID*, *Granularity*, *Factor*) inserts the spatial granularity of the *Geometry\_Name* column in *Table\_Name* relation concerning a *DRS*

which is identified by `DRS_ID`. The *Factor* parameter indicates the conversion rate relative to the unit by default supported by DRS. □

This procedure defines the spatial granularity in a geometry column when the DRS has a unit different from the default one. The SG inserts a new row in the *Spatial\_Ref\_Sys* view.

*Example 2:* Consider example 1. Suppose that the system, defined as `DRS_ID = 14` with ‘Meters’ as default unit, is a DRS. To define the Extension granularity in hectometers, we should use the SG() procedure with the following parameters.

```
SG ('City', 'Extension', 14, 'Hms', 100.0)
```

The last parameter indicates the mapping between meters and hectometers. □

Another important aspect is to provide functions to compare geometries with different spatial granularities. Two geometries with different spatial granularities can be compared if they have associated conversion functions that modify their granularity while maintaining their topological properties. Thus, we must extend the set of functions defined in [29] to support the conversion functions and to create a view in the *SOR-M* recording how to apply them.

We extend the *SOR* model by adding the conversion functions shown in Table 1 as SQL3 functions (Table 3). *SOR-M* is extended to also include a view called *S\_Coarser* that specifies what conversion functions can be applied to a geometry column.

**Definition 6:** Let  $R(A_1:D_1, \dots, A_n:D_n)$  be a *feature relation*,  $\forall A_i (i \in \{1, \dots, n\}) A_i \in G$ , we can define the conversion functions for  $A_i$  by modifying *S\_Coarser* view with the SC procedure, defined as follows:

```
SC(Table_Name, Geometry_Column, Order, Sg, Eg, Operator_Name, Condition)
```

The *Order* attribute indicates the application sequence of the operator the name of which is stored in *Name\_Operator*. This attribute is needed when a spatial data has conversion functions concatenation to specify the change to several coarser granularities. The *Sg* and *Eg* record the initial and final granularities necessary to apply the conversion function and *Condition* is an expression that indicates what objects are affected for this operator. □

**Definition 7:** Let  $R(A_1:D_1, \dots, A_n:D_n)$  be a *feature relation* in the *SOR* model. We define *SGranularity (Name\_Attribute, Spatial\_Ganularity): Geometry* such that *SGranularity()* is a SQL3 function transforming the *Name\_Attribute* spatial column values to values in the indicated granularity by *Spatial\_Ganularity* according to the specification in *S\_Coarser* view. □

*Example 3:* Following with the example 2, we can specify that the *Extension* attribute can change to a coarser granularity through application of the *r\_thinning* operator. The conversion function for this attribute is specified in the meta-scheme through the SC procedure the signature of which is the following:

```
SC ('City', 'Extension', 1, 'Hms', 'Kms', 'r_thinning', ALL);
```

As such signature specification shows, the *Extension* column can change the hectometers to kilometers granularity by applying the *r\_thinning* function over all

*Extension* geometry. In this case, we are indicating that *Extension* values can be represented at kilometers as well. The *Order* attribute records the *I* value although in this case is not significant because we just apply one conversion operator.

The city names and their extensions in kilometers can be queried as:

```
Select SGranularity (Extension, Kms)
From City;
```

□

**Table 2.** Functions of spatial conversions

<b>Contract functions</b>	l_contr (G:Line): Point
	r_contr (G: Polygon): Point
	r_thinning (G: Polygon): Line
<b>Merge functions</b>	l_merge (L1: Line, L2: Line): Line
	r_merge (G1:Polygon, G2:Polygon):Polygon
<b>Absorption operations</b>	p_abs (G:Polygon, P:Point):Polygon
	l_abs (G:Polygon, L:Line):Polygon

The extended *SOR* model provides two important semantics restrictions. First, the spatial granularity definition through the *SR* procedure defines a new spatial reference system, and the *SG* procedure changes the granularity in a default spatial reference system. The second is the facility supporting the representation of a geometry type in several spatial granularities using the conversion functions.

## 5 Temporal Representation in the *SOR* Model

In the object relational model, the temporal representation can use a point-based approach or an interval-point approach and it could affect the relation’s tuples, called tuple level [6], [22], [23], [8] or to the relation attribute, attribute level [13], [14]. As we discussed in the Section 2, the point-based approach avoids the coalescing problems and provides a simpler vision of spatial-temporal systems. For this reason, our proposal represents the time with a point-based approach. Furthermore, it focuses on attribute level because we believe that is closer to the real world and supports the definition of several temporal attributes in the same relation.

We consider a temporal granularity as *foreseeable* or *unforeseeable*. A foreseeable granularity describes periodical phenomena. Granules are calculated through a time function and a temporal seed. The foreseeable temporal granularity can be calculated by applying the time function to the seed, e.g. if the seed is ‘12-1-2003’, and the time function is to add one day, the ‘13-2-2003’ records the next timestamp. The temporal seed can be the first value observed or a value chosen by the user. The unforeseeable granularity is described by random phenomena. For example, if we want to represent the price changes of shares, we could define the unforeseeable granularity in that we do not know the time units when the share could change. This temporal granularity classification allows us to differentiate among several application domains and will be

represented in our approach by the definition of a temporal reference system within the Gregorian calendar.

The proposal will carry out a temporal representation with the definition of abstract data types provided by the standard SQL3 inside the spatial framework explained before. The new data types will take into account that the temporal dimension can be expressed as a function over time for a spatial or non-spatial attribute. This function will be provided by the DBMS or/and customized by the user to collect the information for a certain domain.

In the next section, we explain the approach we adopt to represent temporal attributes in the *SOR* model; specifically; we focus on the presentation of the *foreseeable object moving* representation.

### 5.1 Temporal Attributes

Until now, we can classify the relation columns as *atomic* or *spatial* depending on the data type over which they are defined. The atomic columns are defined over a predefined SQL3 data type and the spatial columns over *Geometry* type. Both types could be grouped and denoted as non-temporal columns (*NTD*). The *temporal column* will be considered as an extension of the atomic and spatial column where the changes produced along the time in those columns will be reflected. At present, the spatial or non-spatial representation shows the current values of the application domain. In this section, we describe how the time evolution is presented.

Let *SOR* be the spatial relational model. The *spatio-temporal object relational* model (*TSOR*) is defined as an extension of *SOR* where the data types are specialized by adding a time attribute. This model is supported by the meta-scheme *TSOR-M*.

**Definition 8:** The  $TSOR-M = \{TM_1, \dots, TM_n\}$  is a set of relations that describe the temporal domain. We define  $V_T$  as a set of views,  $V_T = \{V_{T1}, \dots, V_{Tn}\}$  such that  $V_{Tj} = \text{Opt} = (TM_k, \dots, TM_j)$  where  $\text{Opt}$  is a macro-operator composed of relational algebra operators. □

By using such model extension, we can define the temporal data type as follows.

**Definition 9:** Let  $\alpha$  be a data type belonging to *NTD*, the temporal data type based on  $\alpha$  is denoted by  $T\alpha$  and it is defined as a pair (*Tvalue*, *Tgranule*) where the *Tvalue* is a valid value over  $\alpha$  data type in the *Tgranule* time instant, such that  $Tgranule \in S$  where  $S \in G_T$  and  $S$  is the temporal granularity for  $T\alpha$ . □

In order to support attribute evolution we use the collection type defined in SQL 3.

**Definition 10:** Let  $R(A_1:D_1, \dots, A_n:D_n)$  be a relation  $R$ .  $R$  is a *temporal relation* if  $\exists A_i$  ( $i \in \{1, \dots, n\}$ ) such that  $A_i \in \text{Collection\_}T\alpha$  and  $\text{Collection\_}T\alpha$  records a collection of ordered pairs of  $T\alpha$  type. Each  $A_i \in \text{Collection\_}T\alpha$  is denoted as *temporal column*.

A particular case is when  $\alpha$  is the  $G$  spatial data type.  $R$  is a *temporal -feature* and  $\forall A_i$  ( $i \in \{1, \dots, n\}$ )  $A_i \in \text{Collection\_}TG$ ,  $A_i$  is denoted as *moving column*. □

The temporal column definition in a relation requires that the meta-scheme, the *Temporal\_Columns* view to be exact, be updated in order to maintain the data consistent.

Therefore, a temporal granularity is defined for each temporal column. The granularity is defined by using a SQL3 procedure, like the spatial granularity procedure.

**Definition 11:** Let  $TSOR$  model and  $R(A_1:D_1, \dots, A_n:D_n)$  a temporal relation. The temporal granularity  $\forall A_i (i \in \{1, \dots, n\})$  such that  $A_i \in \text{Collection\_T}\alpha$  is defined by  $TG$ .  $TG$  is a SQL3 procedure which modifies the *Temporal\_Columns* view in  $TSOR\_M$  inserting a new row with the following specifications:

*TG (Table\_Name, Temporal\_Column, Unit, Granularity, Seed, Rate, Function)*

The  $TG$  signature specifies various information. *Unit* is the temporal domain that can get the values Y (year), M (month), D (day), H (hour), Mi (minutes) and S (second), all of them belonging to SQL3 DATETIME. *Granularity* describes the chosen temporal unit and *Rate* denotes the conversion factor to obtain the next granule from *Seed*. When the time function is different from an arithmetic succession of the time, the *Function* attribute stores the user-defined function name.  $\square$

According to this definition, we have covered the *foreseeable temporal* granularities to represent applications where certain phenomenon is observed at periodic timestamp.

Moreover, we create the conversion functions described in Table 1 as SQL3 functions to facilitate the integration and comparison among attributes with different temporal granularities. The  $SOR\_M$  is extended by including a view denoting  $T\_Coarser$  that specify what conversion functions are applied to each temporal column.

**Definition 12:** Let  $R(A_1:D_1, \dots, A_n:D_n)$  be a temporal relation,  $\forall A_i (i \in \{1, \dots, n\})$  such what  $A_i \in \text{Collection\_T}\alpha$ , we can define the conversion specification for  $A_i$  modifying  $T\_Coarser$  view with the  $TG$  procedure. The  $TG$  signature is:

*TG (Table\_Name, Temporal\_Column, Order, St, Et, Operator\_Name, Condition)*

where, the *Order* attribute indicates the application sequence of the operator whose name is stored in *Name\_Operator*. The *St* and *Et* record the initial and final granularity parameters necessary to apply the conversion function and *Condition* is an expression that indicates which objects are affected by this operator.  $\square$

**Definition 13:** Let  $R(A_1:D_1, \dots, A_n:D_n)$  be a temporal relation in the  $TSOR$  model. We define  $TGranularity (Name\_Attribute, Temporal\_Granularity):\text{Collection\_T}\alpha$  as a SQL3 functions that transform the temporal column values,  $Name\_Attribute \in \text{T}\alpha$ , to values in the indicated temporal granularity,  $Temporal\_Granularity$ , according to  $T\_Coarser$  view.  $\square$

Therefore, the  $TG()$  procedure and  $TGranularity()$  contribute to extend the  $TSOR$  model.

*Example 4:* In this example, *City* is a temporal feature relation (see Fig. 3) because the *Extension* is defined as a moving column.

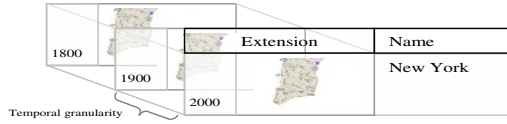
```
City(Name:varchar(60), Extension:Collection_Tpolygon)
TG('City', 'Extension', 'Y', 'Century', 100);
SG('City', 'Extension', 14, 'Hms', 100.0);
```

We use the SG and TG procedures to define the spatio-temporal granularity in Extension and the SC operator to specify the spatial conversion function that can be applied.

```
SC ('City', 'Extension', 1, 'Hms', 'Kms', 'r_thinning',
  ALL);
```

To determine the New York boundary at year 1900, we specify the following query using *SGranularity()* operator.

```
Select SGranularity (c.Extension[].Tvalues, 'Kms') From
City c Where c.Name='New York' and
c.Extension[].Tgranule = '1900';
```



**Fig. 3.** Example to illustrate the City temporal feature relation

The *SGranularity()* realizes a query to *S\_Coarser* view description to calculate the correct topology by the geometry. The *Extension* column changes from hectometers to kilometers granularity applying *r\_thinning* function. □

Each temporal attribute has defined a temporal granularity and the relation is the context that joins them. For this reason, the database design aspects are more significant; we must choose the temporal attributes, in a relation, with granularities that define the domain semantics. With insertion, delete or update operations we must check the temporal validity according to the definition in the *TSOR-M*. Therefore, the temporal validity maintenance could be achieved through the temporal information that is included in the meta-scheme. The implementation of active mechanisms will allow to validate the time for each operation in the database. These mechanisms are strongly domain-dependent because describe the time relationships among the attributes of a relation.

The temporal granularity for a temporal column is an inherent restriction of the *TSOR* model. That is, for each temporal attribute, the model forces one to define a temporal granularity. A semantic restriction is defined in the *TSOR* model giving the possibility to describe the conversion functions. Therefore, this model is semantically richer and allows one to support a larger variety of application domains.

## 7 Conclusion and on Going Research

The proposed framework is based on [29] with the definition of the Geometry data type. We have discussed how the spatial granularity is implicitly defined in the spatial reference system described in [29]. At the spatial level, our approach proposed the

following improvements with respect to a SOR model: an extension of the meta-schema described in [29], referred to as *SOR-M*, that includes new views, functions and operators. Among the new views, the *S\_Coarser* view specifies the conversion functions required to change the spatial granularity preserving the consistency of the spatial attribute. The conversion operators are added to the functions presented in [29].

With respect to the temporal level, our approach is the first proposal to represent temporal attributes in the framework described before. We explain and define how the SQL3 data types can be converted into SQL3 temporal data types. This proposal is based on [13] but our temporal treatment is addressed to point-based approach. The temporal representation based on points makes it easy the use of aggregate functions as proposed in [28]. We define, as well, new views, functions and procedures extending the SQL3 functionalities. These functionalities are necessary to keep the consistency between temporal attributes in a relation and to deal with different granularities.

The main problem when we apply the relational model is the bi-dimensional structure of the relations. This problem reverts in our approximation and generates some difficulties that we are currently investigating. In a relation with several temporal attributes, the attribute with the finest granularity will be the marker of the variability in the relation. To improve the storage problems we are investigating the use of other SQL3 structures, as the REF type in order to avoid redundant information and the use of point-based view at the query level and the interval-based view at the storage level following the approach reported in [25]. The consistency of the temporal attributes within the same relation is another problem that in this approach has been commented. As future work, we plan to develop a set of triggers to check the time validity between attributes of the same relation as well as the execution model and its implications on the inheritance relationships among object types. Finally, other future work will dealing with supporting granularities per value instead of per column like the TOOBIS project approach [31] and the development of an effective and efficient query processing technique. This is an important issue to improve the retrieval of data from temporal attributes using new indexing techniques.

## References

1. Bertolotto, M. Geometric Modeling of Spatial Entities at Multiple Levels of Resolution. PhD Thesis, Università degli Studi di Genova (1998).
2. Bettini, C., Jajodia, S. & Wang, X. Time Granularities in Databases, Data Mining and Temporal Reasoning. Springer-Verlag (2000).
3. T. Bittner and B. Smith. A Unified Theory of Granularity, Vagueness and Approximation. In Proc. of COSIT Workshop on Spatial Vagueness, Uncertain and Granularity (2001).
4. Camossi, E., Bertolotto, M., Bertino, E., Guerrini, G. A multigranular Spatio-temporal Data Model. Proc. 11th ACM International Symposium on Advances in Geographic Information Systems, New Orleans, Louisiana, USA (1998) 94 – 101.
5. Codd, E. F. A Relational Model of Data for Large Shared Data Banks. CACM 13 (6), June (1970).
6. Chen, C.X. & Zaniolo, C. SQL<sup>st</sup>: A Spatio-Temporal Data Model and Query Language. In Proc. of Int. Conference on Conceptual Modeling/ The Entity Relational Approach (2000).



7. Cindy X. Chen, Jiejun Kong and Carlo Zaniolo. Design and Implementation of a Temporal Extension of SQL. In Proceedings of the 19th International Conference on Data Engineering (ICDE'03), pages 689-691, Bangalore, India, March (2003).
8. Cindy Chen, Haixun Wang, and Carlo Zaniolo. Toward Extensible Spatio-Temporal Databases: an approach based on User-Defined Aggregates. In Flexible querying and reasoning in spatio-temporal databases: theory and applications, Springer Geosciences/Geoinformation series (2004).
9. Date, C. J. An Introduction to Database Systems. 8th Edition. Addison-Wesley, Reading, Mass (2003).
10. Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.E., Zemke, F. SQL: 2003 Has Been Published. SIGMOD Record, vol. 33, no. 1, March (2004).
11. Elmasri, R. and Navathe, S: Fundamentals of Database Systems. Addison-Wesley, (2004).
12. Erwig, M., Schneider, M. & Güting R. H. Temporal Objects for Spatio-Temporal Data Models and a Comparison of Their Representations. ER Workshop (1998).
13. Erwig, M., Güting R. H., Schneider, M. & Vazirgiannis, M. Abstract and Discrete Modeling of Spatio-Temporal Data Types. In Proceedings of the 6th ACM Symposium on Geographic Information Systems, pg. 131-136, Washington, D.C., Novembre (1998).
14. Forlizzi, L., Güting, R.H., Nardelli, E., Schneider, M. A data model and data structures for moving objects database. Proceedings of the 2000 ACM SIGMOD international conference on Management of data (2000) 319-330.
15. Güting, R.H. An Introduction to Spatial Database Systems. VLDB Journal, vol. 3, (1994) 357-399.
16. R.H. Güting, M.H. Böhlen, M. Erwig, C.S. Jensen, N.A. Lorentzos, M. Schneider and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. ACM Transactions on Database Systems, Vol.25, No.1 (2000).
17. V. Katri, S. Ram, R.T. Snodgrass and G. O'Brien. Supporting User Defined Granularities and Indeterminacy in a Spatiotemporal Conceptual Model. Special Issues of Annals of Mathematics and Artificial Intelligence on Spatial and Temporal Granularity, 36(1-2) (2002) 195-232.
18. Lee, Y.L. Integrating Spatial and Temporal Relationship Operators into SQL 3 for Historical Data Management. ETRI Journal, vol. 24, n. 3, June (2002).
19. Lorentzos N.A. and Mitsopoulos Y.G. SQL Extension for Interval Data. IEEE Trans. on Knowledge and Data Engineering, vol. 9, no. 3, May/June (1997) 480-499.
20. Muller, J.C, Lagrange, J.P. & Weidel, R. (eds). GIS and Generalization: methodology and practice. Taylor and Francis (1991).
21. OMG (2000). Unified Modelling Language Specification, Version 1-3. Object Management Group.
22. Snodgrass, R.T. The TSQL2 Temporal Query Language. The TSQL2 Language Design Committee, Kluwer Academic Publishers (1995).
23. Snodgrass, R. T. Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann Publishers, Inc., San Francisco, July (1999).
24. Schneider, M. Spatial Data Types for Database Systems-Finite Resolution Geometry for Geographic Information Systems. LNCS 1288, Springer-Verlag (1997).
25. Toman, D. A Point-Based Temporal Extension of SQL. In Proceedings of the 6<sup>th</sup> International Conference on Deductive and Object-Oriented Databases (1997)103-121.
26. Worboys, M. A Unified Model for Spacial and Temporal Information. The Computer Journal, 37(1) (1994) 26-34.
27. Wang, H. & Zaniolo, C. User Defined Aggregates in Object-Relational Systems. In 16th International Conference on Data Engineering, Feb (2000).

28. Wang, H. & Zaniolo, C. Using SQL to Build New Aggregates and Extenders for Object Relational Model. Proc. of the 26<sup>th</sup> International Conference a Very Large Databases (2000) 166-175.
29. [www.opengis.org](http://www.opengis.org)
30. [www.oracle.com](http://www.oracle.com)
31. [www.mm.di.uoa.gr/~toobis/](http://www.mm.di.uoa.gr/~toobis/)

# Managing Inheritance Hierarchies in Object/Relational Mapping Tools

Luca Cabibbo and Antonio Carosi

Dipartimento di Informatica e Automazione,  
Università degli studi Roma Tre

**Abstract.** We study, in the context of object/relational mapping tools, the problem of describing mappings between inheritance hierarchies and relational schemas. To this end, we introduce a novel mapping model, called  $M^2ORM^2+HIE$ , and investigate its mapping capabilities. We first show that  $M^2ORM^2+HIE$  subsumes three well-know basic representation strategies for mapping a hierarchy to relations. We then show that  $M^2ORM^2+HIE$  also allows expressing further mappings, e.g., where the three basic strategies are applied independently to different parts of a multi-level hierarchy. We describe the semantics of  $M^2ORM^2+HIE$  in term of how CRUD (i.e., Create, Read, Update, and Delete) operations on objects (in a hierarchy) can be translated into operations over a corresponding relational database. We also investigate correctness conditions.

## 1 Introduction

Enterprise applications are often developed using an object-oriented programming language (e.g., Java or C#) and a relational database. In this common case, applications need to load data from the database, create objects to represent this data in main memory, perform computations involving these objects, and store changes to objects back in the database.

ORM tools (or, simply, mapping tools) are frameworks for storing and retrieving persistent objects; their goal is to support the complex activity of managing the connections between objects and a relational database. ORM tools allow the programmer to manage the persistence of objects by means of standard API's, such as the JDO [11] or the ODMG ones [7], that is, the same way he would use objects in an object database. Persistence is transparent to the programmer, since he does not know actual implementation details. The bridge between objects and underlying relations is realized on the basis of a data mapping specification.

The *meet in the middle* approach is a mapping strategy for ORM tools. It assumes that data classes (e.g., classes in the application logic holding persistent data) and a relational database have been developed in an independent way. The developer should also describe the correspondences between data classes and the relational database. These correspondences describe a “meet in the middle” between the object schema and the relational schema; they are used by the ORM

tool to let objects persist by means of the database. The meet-in-the-middle approach is very versatile, since modifications in data classes and/or in the relational database can be managed by simply redefining the correspondences. There are several object/relational mapping tools offering the meet-in-the-middle approach (e.g., [10, 13, 15, 16]). However, current systems support the meet-in-the-middle approach still in a limited way, especially because they allow defining only rather restricted kinds of correspondences.

A main limitation imposed by current ORM tools concerns the representation of inheritance hierarchies. There are three well known main strategies to represent a class with its subclasses in a relational schema [2, 9]: (i) by using a single relation; (ii) a relation for each class; and (iii) a relation for each concrete (sub-)class (especially if the superclass is abstract). In practice, current ORM tools do not always offer all the three basic representation strategies for inheritance hierarchies. Furthermore, they usually permit to select, for each hierarchy, a single representation strategy, to be applied to the whole hierarchy.

In previous work [5, 6], we have introduced  $M^2ORM^2$  (Mapping<sup>2</sup> Object Relational Mapping<sup>2</sup>), a model to describe mappings between object schemas and relational schemas, to support the transparent management of object persistence based on the meet-in-the-middle approach. In  $M^2ORM^2$  it is possible to express complex correspondences between groups of related classes and groups of related relations; it is also possible to express correspondences describing relationships between groups. It turns out that  $M^2ORM^2$  generalizes and extends the kinds of correspondences managed by current proposals and systems, e.g., [10, 11, 13, 15, 16], thus allowing for more possibilities to meet schemas. However, until now, we did not take account of inheritance hierarchies in  $M^2ORM^2$ .

In this paper, we study, in the context provided by  $M^2ORM^2$ , the problem of establishing mappings between inheritance hierarchies and relational schemas. To this end, we introduce a novel mapping model, called  $M^2ORM^2+$ , and investigate its mapping capabilities. We show that  $M^2ORM^2+$  subsumes the three above cited basic representation strategies for mapping hierarchies to relations. Moreover, we show that it also allows expressing further mappings, e.g., where the three basic strategies are applied independently to different parts of a multi-level hierarchy. We present the structure and semantics of  $M^2ORM^2+$  mappings; more specifically, we describe how CRUD (i.e., Create, Read, Update, and Delete) operations on objects in an inheritance hierarchy can be translated into operations over a corresponding relational database. We also discuss the problem of verifying the correctness of  $M^2ORM^2+$  mappings involving hierarchies. Again, it turns out that  $M^2ORM^2+$  generalizes and extends the kinds of correspondences managed by current systems.

The paper is organized as follows. Section 2 recalls some preliminary notions, including the  $M^2ORM^2$  mapping model and the three basic representation strategies for inheritance hierarchies. Section 3 introduces  $M^2ORM^2+$ , together with a number of examples. The semantics of  $M^2ORM^2+$  is described in Section 4. Section 5 discusses conditions for the correctness of  $M^2ORM^2+$

mappings. Finally, Section 6 discusses related work and Section 7 draws some conclusions.

## 2 Preliminaries

In this section we briefly present the data models and the terminology used in this paper. We also describe basic notions concerning object/relational mappings based on the meet-in-the-middle approach. Finally, we recall three basic representation strategies for representing hierarchies by means of relations.

### 2.1 Data Models

We consider a simple but realistic model, similar to that of UML [4] and ODMG [7]. We focus on the structural features of the model.

A class is a set of objects having the same structural (and behavioral) properties. Each class has a set of attributes associated with it. In this paper we make the simplifying hypothesis that class attributes are of a same simple type, e.g., strings. A key class is a class in which each object can be identified on the basis of the value of one of its attributes, called the key of the class.

An association is a binary relation between a pair of classes, whose instances are between pairs of objects belonging to the two classes. We consider navigability and multiplicity of (roles of) associations.

Classes are also related by inheritance, generalization, specialization. This relationship is also called inheritance in object-oriented programming, since it implies attribute, association, and method inheritance from the superclass to the subclass. We consider single inheritance only. The inheritance relationship induces a hierarchy (or, simply, inheritance) on classes. A hierarchy is a (maximal) rooted tree of classes connected by inheritance relationships. Because of inheritance, an object may belong to multiple classes; indeed, if an object belongs to a class  $C$ , it belongs to all the superclasses of  $C$  as well. However, as it is customary in object-oriented programming, we assume that each object  $o$  has a unique most specific class, that is, a class  $C$  such that  $o$  belongs to  $C$  and the superclasses of  $C$ . In a hierarchy, a class  $C$  is concrete if every object that belongs to  $C$  must also belong to some subclass of  $C$ , that is, if there cannot be objects whose most specific class is  $C$ . By contrast, a class that is not abstract is called abstract. In hierarchies, leaf classes should be concrete.

An object schema is a set of classes, together with associations and inheritance relationships among such classes. Figure 1 shows a sample object schema, comprising a hierarchy. Constraint  $\{ \dots \}$  denotes key attributes.

In the relational model [8], a relation is a set of tuples. We assume that all relation attributes are of a simple type, e.g., strings. A relation schema is a set of relation schemas. At the instance level, a relation is a set of tuples over the attributes of the relation.

We consider the following integrity constraints. Attributes can be or not be key attributes. Each relation has a primary key (or, simply, key). A foreign key is

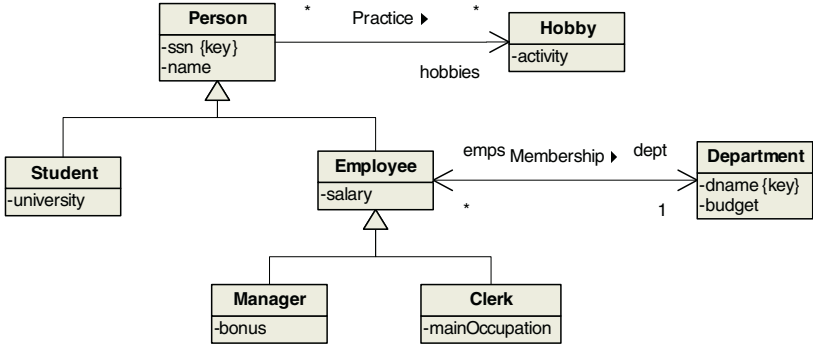


Fig. 1. An object schema

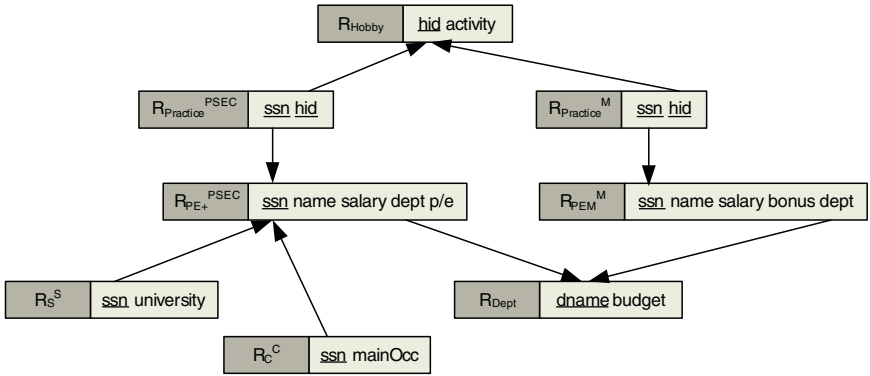


Fig. 2. A relational schema

an attribute that belongs to a key; key attributes must be non null. A  $R_{i,j}$  is a non-empty set of attributes of a relation used to reference tuples of another relation; foreign keys define dependencies between relations. Figure 2 shows a sample relational schema. Attributes forming primary keys are underlined. Referential constraints are denoted by arrows.

## 2.2 Object/Relational Mappings

When an ORM tool is used, objects and links are manipulated by means of operations (create, read, modify, delete), which allow the programmer to create persistent objects, to read persistent objects (that is, perform a unique search of an object based on its key), as well as to modify and delete persistent objects. Navigation, formation, breaking, and modification of persistent links between objects are also possible. In correspondence to such programmatic manipulations of an object schema, an ORM tool should translate CRUD operations on objects and links into operations over the

underlying relational database. This translation should happen in an automatic way, on the basis of a suitable mapping between the object schema and the relational schema, as described next and in the following of this paper.

We now briefly describe the M<sup>2</sup>ORM<sup>2</sup> mapping model [5, 6]. (We refer the reader to our previous work on M<sup>2</sup>ORM<sup>2</sup> for a more detailed presentation of this mapping model.) For the sake of presentation, we assume here that the object schema does not contain hierarchies. (This limitation will be removed next.) In M<sup>2</sup>ORM<sup>2</sup>, a mapping between an object schema and a relational schema is represented as a graph, comprising nodes and arcs. A node describes the correspondences between one or more classes and one or more relations. Usually, a node contains just one class; if there are more than one, a class is selected as the primary class of the node and it is related to other (secondary) classes in the node by associations. Similarly, a node contains usually just one relation; if there are more than one, a relation is selected as the primary relation of the node and it is related to other (secondary) relations in the node by referential constraints. The goal of a node is to describe how to represent an object of the primary class (and possibly further related objects from secondary classes) by means of a tuple in the primary relation (and possibly further related tuples in secondary relations). In a node, data (values) flow between objects and tuples as described by associations, each relating a class attribute to a relation attribute. A mapping can also contain arcs, each describing the correspondences between a pair of nodes by relating an association (between the primary classes of the two nodes) and one or more referential constraints (involving the primary relations of the nodes, and possibly others). The semantics of M<sup>2</sup>ORM<sup>2</sup> is described in Section 4.1.

### 2.3 Basic Strategies for Mapping Inheritance Hierarchies

The problem of mapping a hierarchy to a set of relations is described in several textbooks (e.g., [1, 2, 9]) where, among others, three main basic representation strategies are considered. We now describe these strategies, and exemplify their application to the simple inheritance hierarchy shown in Fig. 3. In giving names to relations used to represent hierarchies, we write  $R_S^T$  to denote the fact that this relation has attributes for classes in the set  $S$  and that it contains a tuple for each object whose most specific class is a class in the set  $T$ . We also write  $C \uparrow$  to denote the set comprising a class  $C$  together with its superclasses, and  $C \downarrow$  to denote the set comprising a class  $C$  together with its subclasses. Finally, symbol  $+$  denotes that a primary attribute is used.

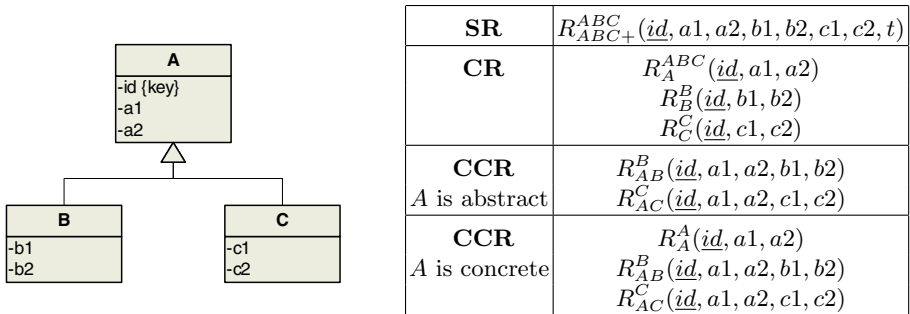
- **Single Relation inheritance (SR):** A hierarchy  $H$  is represented by a single relation  $R_{H+}^H$ . Relation  $R_{H+}^H$  has attributes for all the attributes of classes in  $H$ ; furthermore,  $R_{H+}^H$  has a primary attribute to indicate the most specific class for the object represented by a tuple. An object in the hierarchy is represented by a single tuple in relation  $R_{H+}^H$ . For the hierarchy of Fig. 3, this leads to a single relation  $R_{ABC+}^{ABC}(\underline{id}, a1, a2, b1, b2, c1, c2, t)$ , where  $t$  is the

type attribute, whose possible values are  $A$ ,  $B$ , or  $C$ . This is called *class relation inheritance* in [9].

- **Class Relation inheritance (CR):** A hierarchy  $H$  is represented by a relation  $R_C^{C\downarrow}$  for each class  $C$  in  $H$ . Each relation  $R_C^{C\downarrow}$  has attributes for the attributes of the class  $C$  it represents; a relation for a subclass also has a foreign key towards the relation for its (direct) superclass in the hierarchy. An object in the hierarchy is represented by multiple tuples: if the most specific class for the object is  $C$ , by a tuple in relation  $R_C^{C\downarrow}$  which represents  $C$ , together with a tuple for each relation  $R_{C'}^{C\downarrow}$  that represents a superclass  $C'$  of  $C$ . For the hierarchy of Fig. 3, this leads to relations  $R_A^{ABC}(\underline{id}, a1, a2)$ ,  $R_B^B(\underline{id}, b1, b2)$ , and  $R_C^C(\underline{id}, c1, c2)$ , where attribute  $id$  in relations  $R_B^B$  and  $R_C^C$  references relation  $R_A^{ABC}$ . This is called *class relation inheritance* in [9].
- **Concrete Class Relation inheritance (CCR):** A hierarchy  $H$  is represented by a relation  $R_{C\uparrow}^C$  for each concrete class  $C$  in  $H$ . Each relation  $R_{C\uparrow}^C$  has attributes for the attributes of the concrete class  $C$  it represents, but also for attributes for each superclass of  $C$ . An object in the hierarchy is represented by a single tuple in the relation  $R_{C\uparrow}^C$  for the most specific class  $C$  of the object. For the hierarchy of Fig. 3, assuming that class  $A$  is abstract, this leads to relations  $R_{AB}^B(\underline{id}, a1, a2, b1, b2)$  and  $R_{AC}^C(\underline{id}, a1, a2, c1, c2)$ . However, if  $A$  is concrete, this leads to relations  $R_A^A(\underline{id}, a1, a2)$ ,  $R_{AB}^B(\underline{id}, a1, a2, b1, b2)$ , and  $R_{AC}^C(\underline{id}, a1, a2, c1, c2)$ . This is called *concrete class relation inheritance* in [9] and *concrete class relation inheritance* in [12].

To the best of our knowledge, current object/relational mapping tools adopt mainly these three basic strategies, even though other representation strategies for inheritance hierarchies are known (e.g., *class relation inheritance* [1]).

The basic representation strategies can be applied to an inheritance hierarchy as a whole. In case of a multi-level hierarchy (e.g., where a subclass can have its own sub-subclasses, etc.), it is also possible to apply different strategies to distinct parts of the hierarchy. A possible advice is to apply the representation procedure recursively, starting from the bottom of the hierarchy and representing one inheritance level at a time [2].



**Fig. 3.** A simple hierarchy and three basic relational representations



In practice, current ORM tools do not always offer all the three above cited basic representation strategies for hierarchies. Furthermore, they usually permit to select, for each hierarchy, a single representation strategy, to be applied to the whole hierarchy. (See Section 6 for a discussion on the management of hierarchies in current tools.) These facts limit the number of possible mappings that can be identified among an object schema with hierarchies and a relational database.

### 3 A Model for Mapping Hierarchies and Relations

We now describe a mapping model for dealing with inheritance hierarchies. Specifically, we extend  $M^2ORM^2$  to represent hierarchies by means of a novel kind of arcs, called inheritance arcs. This extension is called  $M^2ORM^2+$ .

In  $M^2ORM^2+$ , a  $\langle C_2, R_2, C_1, R_1 \rangle$  describes the correspondences between a primary class and a primary relation, possibly involving other classes (related by associations and/or inheritance) and other relations (related by referential constraints).

In nodes, apart from  $\langle C, R \rangle$  (each relating a class attribute with a relation attribute), relation attributes can also be related to constant values by means of  $\langle R, v \rangle$ .

An  $\langle C_2, R_2, C_1, R_1 \rangle$  from a node  $N_2$  to a node  $N_1$  represents an inheritance relationship from the primary class  $C_2$  of  $N_2$  to the primary class  $C_1$  of  $N_1$ , that is, the fact that  $C_2$  is a (direct) subclass of  $C_1$ . Normally, an inheritance arc specifies that attribute and literal correspondences are inherited from the node for the superclass to the node for the subclass. However, correspondences in a node can override inherited correspondences.

An inheritance arc can have an associated  $\langle R_2, R_1 \rangle$ , from a key attribute of the primary relation in the node for the subclass to the key attribute of the primary relation in the node for the superclass. This specifies a referential constraint between the two relations.

In  $M^2ORM^2+$ , some elements can be abstract. An  $\langle C, R \rangle$  contains just an abstract class, but no relations. An abstract node defines, implicitly, an  $\langle C, R \rangle$  for each attribute of the class; intuitively, abstract correspondences specify correspondences that should be provided by nodes where the abstract correspondences are inherited. An abstract node cannot contain “concrete” correspondences. An  $\langle C_2, R_2, C_1, R_1 \rangle$  specifies that the node for the subclass does not inherit correspondences from the node for the superclass; rather, these correspondences should be considered abstract correspondences, and they should therefore be redefined in the node for the subclass. Intuitively, an abstract inheritance arc towards the node for a concrete class is similar to an inheritance arc towards the node for an abstract class.

As in  $M^2ORM^2$ , relationship arcs are also allowed [5, 6].

The following example shows how the mappings implied by the three basic representation strategies of Section 2.3 can be described in  $M^2ORM^2+$ .

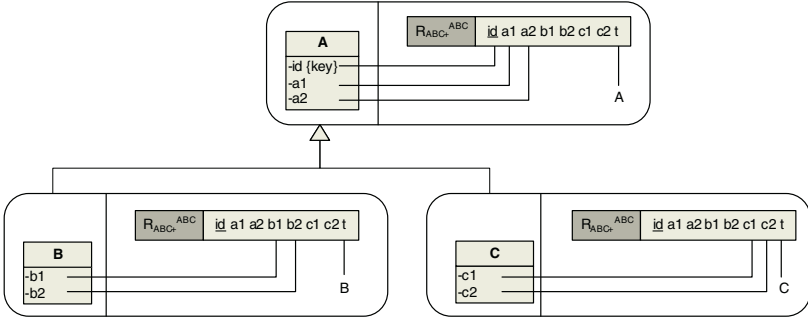


Fig. 4. SR in  $M^2ORM^2+HIE$

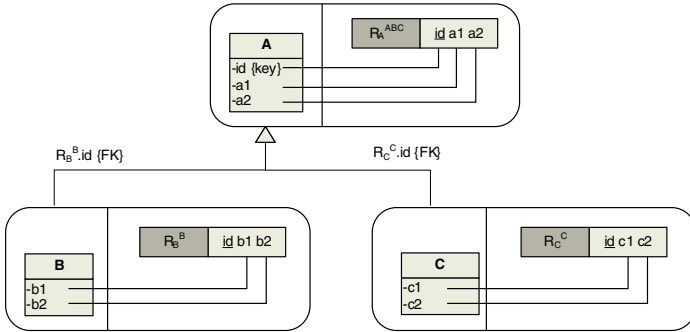


Fig. 5. CR in  $M^2ORM^2+HIE$

Consider the schemas of Fig. 3.

In  $M^2ORM^2+$ , the mapping for the translation implied by can be represented using a node for each class,  $N_A$ ,  $N_B$ , and  $N_C$ , together with inheritance arcs between them, that is, from  $N_B$  to  $N_A$  and from  $N_C$  to  $N_A$ . Each node relates a class of the hierarchy to relation  $R_{ABC+}^{ABC}$ ; moreover, each node has an attribute correspondence for each attribute of the class of the node (relating it to the corresponding relation attribute, e.g.,  $A.a1$  to  $R_{ABC+}^{ABC}.a1$ ). Finally, node  $N_A$  has literal correspondence  $R_{ABC+}^{ABC}.t = A$ , whereas nodes  $N_B$  and  $N_C$  override it as  $R_{ABC+}^{ABC}.t = B$  and  $R_{ABC+}^{ABC}.t = C$ , respectively.<sup>1</sup> Figure 4 shows this mapping.

The mapping for can be represented, again, by using three nodes and two arcs, as it is shown in Fig. 5. Each node relates a class of the hierarchy to the corresponding relation, e.g., node  $N_A$  relates class  $A$  to relation  $R_A^A$ ; each node has attribute correspondences for the attributes of the class of the node. In this case, inheritance arcs carry further information needed to complete the

<sup>1</sup> Literal correspondences such as  $R_{ABC+}^{ABC}.b1 = null$  in  $N_A$  are not needed, since this is the default in  $M^2ORM^2$ .

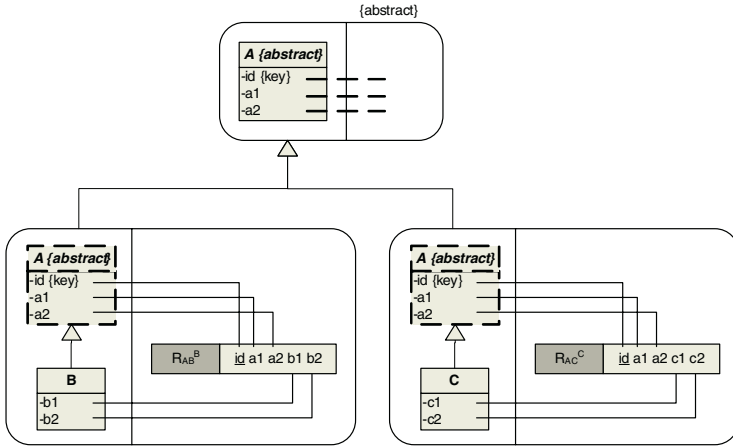


Fig. 6. CCR in  $M^2ORM^2+HIE$  (class  $A$  is abstract)

mapping specification; in particular, the inheritance arc from  $N_B$  to  $N_A$  holds a foreign key correspondence from  $R_B^B.id$  to  $R_A^{AB}.id$ . The case for the arc from  $N_C$  to  $N_A$  is similar.

Figure 6 shows the mapping for , , when class  $A$  is abstract. Again, three nodes are needed, as well as inheritance arcs between them. Node  $N_A$  is abstract, and as such it has abstract correspondences for attributes of class  $A$ . Nodes  $N_B$  and  $N_C$  relate concrete classes  $B$  and  $C$  to relations  $R_{AB}^B$  and  $R_{AC}^C$ , respectively. Each node for a concrete class has attribute correspondences for the attributes of the class of the node, but also for the attributes of their abstract superclass

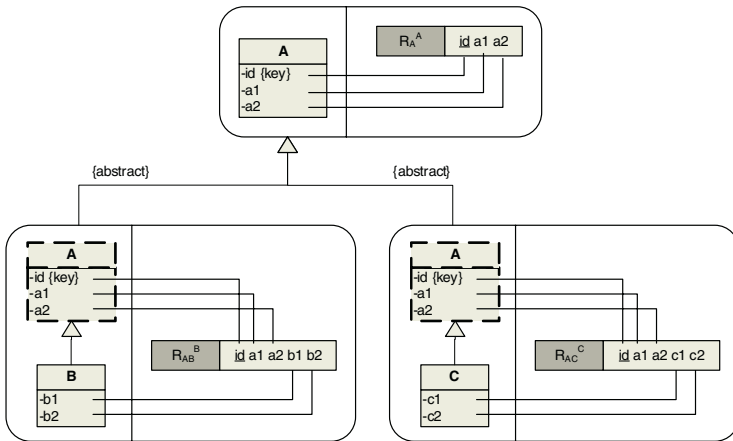


Fig. 7. CCR in  $M^2ORM^2+HIE$  (class  $A$  is concrete)

for which an abstract attribute correspondence is inherited, e.g., node  $N_B$  has an attribute correspondence between  $A.a1$  and  $R_{AB}^B.a1$ .

The mapping for  $\langle \cdot, \cdot \rangle$  when  $A$  is concrete is shown in Fig. 7. Node  $N_A$  relates class  $A$  to relation  $R_A^A$ . The mapping contains also abstract inheritance arcs from  $N_B$  to  $N_A$  and from  $N_C$  to  $N_A$ . Node  $N_B$  relates class  $B$  to relation  $R_{AB}^B$ .  $N_B$  has attribute correspondences for the attributes of  $B$ , but also for the attributes of its superclass  $A$ , e.g., node  $N_B$  has an attribute correspondence between  $A.a1$  and  $R_{AB}^B.a1$ . The case for node  $N_C$  is similar.  $\square$

Besides mappings corresponding to the three basic representation strategies for hierarchies described in Section 2.3,  $M^2ORM^2+$  allows specifying more complex mappings, as the following example shows.

Consider the hierarchy in the object schema of Fig. 1. Assume all the classes are concrete. Figure 8 shows a complex mapping between this hierarchy and the relational schema of Fig. 2. (We use letters  $P, S, E, M,$  and  $C$  to denote classes,  $\dots$ ,  $\dots$ ,  $\dots$ ,  $\dots$ , and  $\dots$ , respectively.) Relation  $R_{PE+}^{PSEC}$  has tuples for  $\dots$  s,  $\dots$  s,  $\dots$  s, and  $\dots$  s (but not for  $\dots$  s); relation  $R_S^S$  holds further data for  $\dots$  s, whereas relation  $R_C^C$  holds further data for  $\dots$  s; finally, relation  $R_{PEM}^M$  holds all information for  $\dots$  s. Five nodes are used, together with the three different kinds of arcs

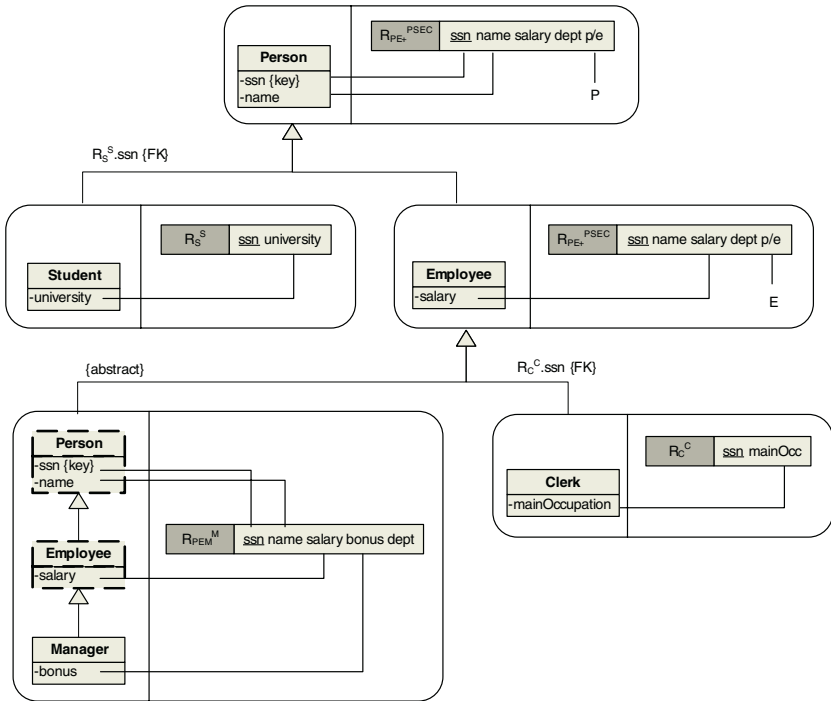


Fig. 8. A complex mapping over the hierarchy of Fig. 1

(i.e., with and without foreign key correspondence, and abstract). We can think of this relational schema not as obtained by applying a single representation strategy to the whole hierarchy, but rather by applying different strategies to distinct parts of the same hierarchy.  $\square$

We would like to point out that, to the best of our understanding, none of the systems we have analyzed (including, among others, [10, 13, 15]) is able to manage a mapping similar to the one described by Example 2.

## 4 Semantics of Mappings

In this section we present the semantics of  $M^2ORM^2+$ . We first briefly recall the semantics of  $M^2ORM^2$  [5, 6] (where inheritance is not allowed).

### 4.1 Semantics of $M^2ORM^2$

In  $M^2ORM^2$ , a node maps a group of classes (a primary class connected by one-to-one or many-to-one associations to further secondary classes) to a group of relations (a primary relation connected by referential constraints to further secondary relations). The goal of a node is to represent an object  $o$  of the primary class, together with objects in secondary classes that are reachable from  $o$  by means of associations represented within the node, as a tuple  $t_o$  in the primary relation, together with tuples in secondary relations that are reachable from  $t_o$  by means of referential constraints represented within the node.

In general, we have the following intuitive semantics for CRUD operations (applied to an object in the primary class of a node): The *insertion* of an object  $o$  in the primary class is managed as the insertion into the database of tuples (in the primary and in secondary relations) representing both object  $o$  and objects in secondary classes that are reachable from  $o$ ; values flow from object (i.e., class) attributes to tuple (i.e., relation) attributes. To *delete* an object of the primary class, given its key, a query over the database is executed to retrieve the tuples (in the primary and in secondary relations) that represent an object  $o$  in the primary class and objects in secondary classes that are reachable from  $o$ ; then, the corresponding objects are created in memory; values flow from tuple attributes to object attributes. The *update* of attributes of an object (or the update of links between them, thereof) is managed by modifying the tuples used to represent the group of objects. The *deletion* of an object  $o$  in the primary class is managed by deleting the tuple  $t_o$  in the primary relation used to represent object  $o$ .

We do not describe the semantics of association arcs, since its knowledge is not needed here.

### 4.2 Semantics of $M^2ORM^2+HIE$

Before defining the semantics of  $M^2ORM^2+$ , it is worth to note that, in a mapping, a relation may be used to contain tuples representing objects from

$R_{PE+}^{PSEC}$	<table border="1"> <thead> <tr> <th><i>ssn</i></th> <th><i>name</i></th> <th><i>salary</i></th> <th><i>dept</i></th> <th><i>p/e</i></th> </tr> </thead> <tbody> <tr> <td>1234</td> <td>Paul</td> <td><i>null</i></td> <td>...</td> <td>P</td> </tr> <tr> <td>5678</td> <td>Sarah</td> <td><i>null</i></td> <td>...</td> <td>P</td> </tr> <tr> <td>9753</td> <td>Ella</td> <td>14K</td> <td>...</td> <td>E</td> </tr> <tr> <td>8642</td> <td>Charles</td> <td>15K</td> <td>...</td> <td>E</td> </tr> </tbody> </table>	<i>ssn</i>	<i>name</i>	<i>salary</i>	<i>dept</i>	<i>p/e</i>	1234	Paul	<i>null</i>	...	P	5678	Sarah	<i>null</i>	...	P	9753	Ella	14K	...	E	8642	Charles	15K	...	E
<i>ssn</i>	<i>name</i>	<i>salary</i>	<i>dept</i>	<i>p/e</i>																						
1234	Paul	<i>null</i>	...	P																						
5678	Sarah	<i>null</i>	...	P																						
9753	Ella	14K	...	E																						
8642	Charles	15K	...	E																						

$R_S^S$	<table border="1"> <thead> <tr> <th><i>ssn</i></th> <th><i>university</i></th> </tr> </thead> <tbody> <tr> <td>5678</td> <td>Stanford</td> </tr> </tbody> </table>	<i>ssn</i>	<i>university</i>	5678	Stanford
<i>ssn</i>	<i>university</i>				
5678	Stanford				

$R_C^C$	<table border="1"> <thead> <tr> <th><i>ssn</i></th> <th><i>mainOcc</i></th> </tr> </thead> <tbody> <tr> <td>8642</td> <td>archivist</td> </tr> </tbody> </table>	<i>ssn</i>	<i>mainOcc</i>	8642	archivist
<i>ssn</i>	<i>mainOcc</i>				
8642	archivist				

$R_{PEM}^M$	<table border="1"> <thead> <tr> <th><i>ssn</i></th> <th><i>name</i></th> <th><i>salary</i></th> <th><i>bonus</i></th> <th><i>dept</i></th> </tr> </thead> <tbody> <tr> <td>7007</td> <td>Maria</td> <td>25K</td> <td>12K</td> <td>...</td> </tr> </tbody> </table>	<i>ssn</i>	<i>name</i>	<i>salary</i>	<i>bonus</i>	<i>dept</i>	7007	Maria	25K	12K	...
<i>ssn</i>	<i>name</i>	<i>salary</i>	<i>bonus</i>	<i>dept</i>							
7007	Maria	25K	12K	...							

**Fig. 9.** Relations store classes

multiple most specific classes. For example, in the mapping shown in Fig. 8, relation  $R_{PE+}^{PSEC}$  contains a tuple for each object whose most specific class is either  $\dots$ ,  $\dots$ ,  $\dots$ , or  $\dots$ . We say that, in a mapping, a relation  $R_{PE+}^{PSEC}$  stores class  $C$  if it is intended to contain, among others, a tuple for each object whose most specific class is  $C$ . We have the following characterization for the “stores” relationship:

- let  $C$  be a concrete class; the set of relations that  $R_{PE+}^{PSEC}$  stores  $C$  can be computed by visiting the node whose primary class is  $C$  and all the nodes that can be reached from it by climbing up inheritance arcs that are not abstract;
- let  $R$  be a relation; the set of classes that  $R$  stores can be computed by visiting each node containing  $R$  and all the nodes that can be reached from them by going down inheritance arcs that are not abstract (primary classes only).

Figure 9 shows how relations store classes with respect to the mapping described in Example 2.

We can now define the semantics for CRUD operations (applied to objects in an inheritance hierarchy).

**Creation.** Consider the creation of an object  $o$  in the primary class  $C$  of a node. This class will be the most specific class for the object to be created. Object  $o$  has values for the attributes defined in class  $C$ , but also for attributes defined in superclasses of  $C$ . Object  $o$  will be represented by a tuple for each relation that stores  $C$ ; these relations belong to a path, in the graph describing the mapping, from a node for some superclass  $C'$  of  $C$  to the node for  $C$ . Values flow from attributes of  $o$  to tuples representing  $o$ , as described by attribute and literal correspondences. Specifically, attribute and literal correspondences are applied, in sequence and downwards, from the node for  $C'$  to the node for  $C$ . Tuples that are identified in this way are inserted into the database.

For example, consider the mapping described by Example 2. The creation of an object in class  $\dots$  would involve relations  $R_{PE+}^{PSEC}$  and  $R_C^C$ , which store  $\dots$ . The node for  $\dots$  specifies the value for attributes  $\dots$  and  $\dots$  in  $R_{PE+}^{PSEC}$ . In the same relation, the node for  $\dots$  specifies the value for attributes  $\dots$  and  $\dots$  (the literal correspondence in this node overrides the

one in the node for  $(C, \dots)$ . Finally, the node for  $(C, \dots)$  specifies the value for the tuple to be inserted in relation  $R_C^C$ .

**Reading.** Consider the reading of an object from a class  $C$ , given its key  $id$ . When performing this operation, the retrieved object  $o$  should belong to the most specific class among  $C$  and the subclasses of  $C$  (this is known as *polymorphic reading*). Therefore, the reading starts by identifying, by issuing a number of database queries  $q_1, q_2, \dots$ , the most specific class  $C'$  for the object  $o$  whose key is  $id$ . Each query  $q_i$  is relative to some concrete class  $C_i$  that is either  $C$  or some subclass of  $C$ , and has the goal of checking whether the database represents an object belonging to  $C_i$  whose key is  $id$ . Query  $q_i$  comprises a join of the relations that store  $C_i$ , with selections for the key  $id$  and for (possibly inherited) literal correspondences in the node for  $C_i$ . The class  $C'$  for the object we are reading is chosen as the most specific class (i.e., the most downwards in the hierarchy) among those to which the object can belong. Then, the reading proceeds as in the standard semantics of M<sup>2</sup>ORM<sup>2</sup>, by performing a query over the relations that store  $C'$  and by creating, in memory, the desired object  $o$ . Values flow from attributes of the retrieved tuples to  $o$  (and possibly other related objects), as described by attribute correspondences.

Consider again the mapping of Example 2. Assume that a (polymorphic) reading over class  $P_{E+}$  has been requested, given the ssn  $id$ . Three queries  $q_e, q_m$ , and  $q_c$  are performed, to check whether an  $P_{E+}$ , a  $P_{E+}$ , and/or a  $P_{E+}$  does exist whose key is  $id$ . For example, query  $q_c$  for  $P_{E+}$  would be:

```
SELECT * FROM R_{PE+}^{PSEC}, R_C^C
WHERE R_{PE+}^{PSEC}.ssn=id AND p/e='E' AND R_{PE+}^{PSEC}.ssn=R_C^C.ssn
```

Assume the result of query  $q_c$  is not empty. In this case, the most specific class for the retrieved object will be  $P_{E+}$ ; moreover, the result of query  $q_c$  will be used as values for the attributes of the retrieved object.

**Update.** An update can be the modification of either an attribute of an object or a link, described by some node. As it is customary in object-oriented programming, we assume that modifying the most specific class for an object is not allowed. In this case, the update of an object  $o$  is performed as stated in the standard semantics of M<sup>2</sup>ORM<sup>2</sup>, that is, by modifying tuples used to represent  $o$ .

**Delete.** The deletion of an object  $o$  is performed by deleting tuples used to represent  $o$ . Again, for this case there is no difference with respect to the standard semantics of M<sup>2</sup>ORM<sup>2</sup>.

## 5 Correctness of Mappings

Correctness is an important aspect of object/relational mappings. Intuitively, a mapping is correct if it supports, in an effective way, the management of CRUD operations on objects and links by means of the underlying relational database.

We now briefly discuss correctness conditions concerning the mapping of inheritance hierarchies. For the sake of presentation, we now make the following assumptions: (i) each class is primary in exactly one node; (ii) each node contains at most one relation. (For a treatment of cases where the above assumptions do not hold we refer the reader to previous work [5, 6].) In this case, correctness of a mapping requires, at least, the following conditions to hold:

- for each concrete class  $C$ , each of the attributes of  $C$  and of its superclasses is mapped to exactly one relation attribute (apart from possible foreign keys), among the relations that store  $C$ ;
- for each relation  $R$  and each class  $C$  stored by  $R$ , each of the attributes of  $R$  is mapped to at most one class attribute;
- key attributes of classes are related to key attributes of relations;
- class attributes that can be null are related to relation attributes that can be null.

The mapping described by Example 2 satisfies these conditions, and indeed it is a correct mapping between (the hierarchy of) the object schema of Fig. 1 and (part of) the relational schema of Fig. 2. We would like to point out that, using other M<sup>2</sup>ORM<sup>2</sup> mapping features, it is possible to define a correct mapping between the whole schemas shown by Fig. 1 and 2.

## 6 Related Work

There are several object/relational mapping tools available today (for a comparison of some tools supporting Java see [10, 15]); some of them also offer the meet-in-the-middle approach (e.g., [10, 15]). A mainstream application of ORM tools is supporting container-managed persistence (CMP) of Entity Beans in J2EE application servers [18]. Major relational DBMS vendors have recently started offering object/relational mapping tools based on the meet-in-the-middle approach, e.g., Oracle AS TopLink [16] and Microsoft ObjectSpaces [13].

In ORM tools, mappings are usually represented by graphs, as we do in M<sup>2</sup>ORM<sup>2</sup>. For example, in TopLink [16] a mapping comprises *nodes* (corresponding to classes) and *relationships* (corresponding to (relationship) arcs). Often, each node relates just a single class to just a single relation. Some systems (e.g., ObjectSpaces [13]) allow expressing more complex mappings between groups of classes and groups of relations, as we do in M<sup>2</sup>ORM<sup>2</sup>.

Most ORM tools take inheritance hierarchies into account. However, they do not always offer all the three basic representation strategies for inheritance hierarchies described in Section 2.3. Furthermore, they usually permit to select, for each hierarchy, a single representation strategy to be applied to the whole hierarchy. For example, ObjectSpaces [13] allows for all three strategies but, to the best of our understanding, they cannot be applied separately to different parts of a single hierarchy. In Hibernate [10], different strategies can be applied to distinct parts of a same hierarchy, but with some limitations: in particular,



if  $C_1$  and  $C_2$  are two direct subclasses of a same class  $C$ , it is not possible to apply  $\text{M}^2\text{ORM}^2+$  to  $C$  and  $C_1$  and, to  $C$  and  $C_2$ . On the other hand,  $\text{M}^2\text{ORM}^2+$  offers more mapping possibilities with respect to object schemas containing hierarchies. Indeed, none of these systems allows expressing the mapping described by Example 2.

The “professional” literature is rich of works on several aspects concerning the implementation of ORM tools. Many contributions on the topic are now available as book chapters (e.g., [1, 9]) or as web resources (e.g., [14]). On the other hand, the scientific literature on ORM tools (e.g., [17]), apart from our previous work [5, 6], is more limited or, simply, outdated by current technology offerings. The notion of mapping used in this paper is inspired from one proposed in the context of model management [3].

## 7 Discussion

In this paper we have introduced  $\text{M}^2\text{ORM}^2+$ , a mapping model for object/relational mapping tools. With respect to our previous work [5, 6], in this paper we have investigated the problem of managing inheritance hierarchies. It turns out that, as other mapping tools,  $\text{M}^2\text{ORM}^2+$  is able to manage the most common representations for hierarchies. However, unlike other available systems and proposals, in  $\text{M}^2\text{ORM}^2+$  it is also possible to represent more complex mappings involving hierarchies, e.g., where the three basic strategies are applied independently to different parts of a multi-level hierarchy.

There are a number of aspects related to ORM tools that we would like to investigate in the context provided by  $\text{M}^2\text{ORM}^2$  and  $\text{M}^2\text{ORM}^2+$ ; these include: data types, multi-attribute keys, complex attributes, polymorphic associations, and cascade semantics. We believe that such features can be introduced in our mapping model in a graceful way.

## References

1. S.W. Ambler. *Agile Database Techniques*. Wiley Publishing, 2003.
2. C. Batini, S. Ceri, and S.B. Navathe. *Conceptual Database Design, an Entity-Relationship Approach*. Benjamin-Cummings, 1992.
3. P.A. Bernstein, A.Y. Halevy, and R.A. Pottinger. A vision for the management of complex models. *ACM Sigmod Record*, 29(4):55–63, 2000.
4. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
5. L. Cabibbo and R. Porcelli.  $\text{M}^2\text{ORM}^2$ : A Model for the Transparent Management of Relationally Persistent Objects. In *International Workshop on Database Programming Languages (DBPL)*, pages 166–178, 2003.
6. L. Cabibbo. Objects Meet Relations: On the Transparent Management of Persistent Objects. In *Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, pages 429–445, 2004.
7. R.G.G. Cattell et al. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.

8. R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 2003.
9. M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003.
10. Hibernate. <http://www.hibernate.org/>.
11. Java Data Objects. <http://www.jdocentral.com>.
12. W. Keller. Mapping Objects to Tables: A Pattern Language. In *European Pattern Languages of Programming Conference (EuroPLOP)*, 1997.
13. Microsoft ObjectSpaces. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnadonet/html/objectspaces.asp>.
14. Object Architects. Patterns for Object/Relational Mapping and Access Layers. <http://www.objectarchitects.de/ObjectArchitects/orpatterns/>.
15. ObJect relational Bridge. <http://db.apache.org/objb/>.
16. Oracle AS TopLink. <http://otn.oracle.com/products/ias/toplink/>.
17. J.A. Orenstein. Supporting retrievals and updates in an object/relational mapping system. *IEEE Bull. on Data Engineering*, 20(1):50–54, 1999.
18. E. Roman. *Mastering Enterprise JavaBeans*. Wiley Publishing, 2002.

# BInXS: A Process for Integration of XML Schemata<sup>\*</sup>

Ronaldo dos Santos Mello<sup>1</sup> and Carlos Alberto Heuser<sup>2</sup>

<sup>1</sup> Universidade Federal de Santa Catarina,  
Depto. de Informatica e Estatistica, Cx. Postal 476,  
Florianopolis, SC, Brasil 88040-900  
ronaldo@inf.ufsc.br

<sup>2</sup> Universidade Federal do Rio Grande do Sul,  
Instituto de Informatica, Cx. Postal 15064,  
Porto Alegre, RS, Brasil 91501-970  
heuser@inf.ufrgs.br

**Abstract.** This paper presents a detailed integration process for XML schemata called BInXS. BInXS adopts a *global-as-view* integration approach that builds a global schema from a set of heterogeneous XML schemata related to a same application domain. This bottom-up approach maps all element and attribute definitions in XML schemata to correspondent concepts at the global schema, allowing access to all data available at the XML sources. The integration process is semi-automatically performed over conceptual representations of the XML schemata, which provides a better understanding of the semantics of the XML data to be unified. A conceptual schema is generated by a set of conversion rules that are applied to a schema definition for XML data. Once this conceptual schema is the result of a meticulous analysis of the XML logical model, it is able to abstract the particularities of semistructured and XML data, like elements with mixed contents and elements with alternative representations. Therefore, the further unification of such conceptual schemata implicitly deals with structural conflicts inherent to semistructured and XML data. In addition, BInXS supports a mapping strategy based on XPath expressions in order to maintain correspondences among global concepts and data at the XML sources.

## 1 Introduction

The XML format has been extensively used to represent data as well as to interchange data among users and applications, specially through the Web [7]. Several application domains, like e-commerce [1, 3] and bibliographic references [2, 4], provides XML information on the Web. Considering such increasing availability of XML data, schema integration mechanisms are required to provide an unified access to several heterogeneous XML sources on the Web related to a same application domain.

An XML data is a semistructured data [8]. Thus, the integration of XML schemata is more complex than the integration of database schemata because semistructured schemata are irregular, allowing the definition of heterogeneous instances in a same

---

<sup>\*</sup> This work was partially supported by CAPES Foundation.

schema. Because of this high heterogeneity, it is difficult to find out semantic correspondences among XML data based on a structural analysis of them, as well as to solve conflicts of data representation in order to perform a unification.

Database schema integration processes usually convert the data models of the heterogeneous databases to a common data model called *canonical model* [10, 17, 31]. This canonical representation abstracts the heterogeneity of the data models, reducing the complexity of the integration activity. Considering the specific integration of XML schemata, there is a choice between: (i) to convert the XML data model to a canonical model that is able to abstract the high structural heterogeneity of each XML schema or; (ii) does not perform such conversion. Alternative (i) requires a conversion process and mappings from one model to the other. However, the complexity of the further integration is reduced. Alternative (ii) does not require the conversion, but has to deal with the complexity inherent to the integration of XML schemata.

Several related work on semistructured or XML schema integration apply alternative (i) [11, 20, 21, 23, 24, 28, 30]. However, their main drawback is that the adopted canonical model does not consider all the particularities of the XML data model. Consequently, they do not deal with some kinds of conflicts that raise when XML schemata are unified, like elements with mixed content (text and structure) and elements with alternative representations.

This paper presents a process for XML schema integration called **BInXS**<sup>1</sup>. BInXS also follows alternative (i), proposing a conceptual canonical representation to a schema for XML data. Such canonical representation results of a detailed analysis not only of the XML data model, but also of XML instances in order to improve the understanding of data semantics. The further schema unification applied on these canonical schemata takes implicitly into consideration the resolution of conflicts related to XML schemata, like the ones exemplified before. The main advantage of such approach is that the integration is applied on a conceptual basis, i.e., on high level and detailed abstractions of XML schemata. A global conceptual schema is generated at the end of the integration process. This global schema is useful in the context of a mediation system [12] that provides access to XML sources on the Web.

This paper is organized as follows. Section 2 gives an overview of the integration process followed by BInXS. Section 3 describes the conversion of an XML schema to a conceptual schema. Section 4 describes how the global schema is defined from the unification of conceptual schemata. Section 5 discusses some related work. Section 6 is dedicated to the conclusion.

## 2 BInXS Overview

BInXS is a *semi-automatic* and *bottom-up* process for semantic integration of XML schemata [25]. It is *semi-automatic* because user intervention is needed in order to validate the semantic intention of data during the integration process. Semantic integration processes are not fully automatic because the definition of a precise meaning for a data

---

<sup>1</sup> **BInXS** is an acronym for **B**ottom-up **I**ntegration of **X**ML **S**chemata.

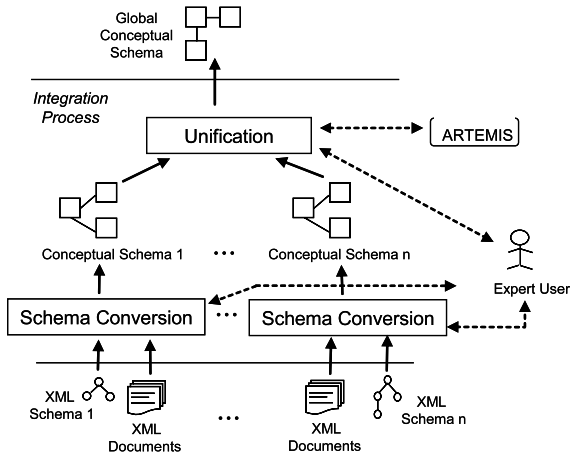


Fig. 1. BInXS integration process

is a very subjective matter. BInXS is also a *bottom-up* process because it generates a global schema from a set of XML schemata, being classified as a *global-as-view* integration approach [18]. Such global schema abstracts the high heterogeneity of the XML data sources and considers the semantic intention of all of these sources.

BInXS has two phases, as shown in Figure 1. The first phase, called *Schema Conversion*, maps each XML logical schema to a correspondent conceptual schema. BInXS adopts a conceptual canonical model because it provides a high level abstraction for the XML data. Besides, a same conceptual schema may abstract several XML logical schemata of a same application domain. The unification of conceptual representations of XML data reduces the complexity of the integration process because it is much simple to find out semantic similarities among conceptual schemata, which straightly represent real world facts and their relationships. The *Schema Conversion* phase is detailed in section 3.

Not only XML schematic information are analyzed in this first phase, but also data in XML documents. Such data analysis is necessary to define a more accurate conceptual schema, helping on the definition of relationship cardinalities and relationships derived from XML element references, for example. The intervention of an expert user is expected to validate automatic-generated conceptual schema concepts in order to obtain a definitive conceptual schema. Mapping information from conceptual schema concepts to XML elements or attributes are also generated and kept in a catalog.

The second phase, called *Unification*, takes a set of conceptual schema generated from the previous phase and performs their semantic integration, creating a global conceptual schema. An external tool, called ARTEMIS, is used to find out semantic affinities between concepts in different schemata. User intervention is considered again to eventually choose one among several alternative semantic meanings for a global concept or relationship representation, or to validate an automatic-generated preliminary global schema. Section 4 details this phase.

### 3 Schema Conversion

The *Schema Conversion* phase is based on a set of rules that consider the concepts of the XML model, analysis of XML documents, and user expertise [27]. The conversion process has three steps: *Pre-processing*, *Conversion* and *Restructuring*.

The *Pre-processing* step takes an XML schema (a DTD or XSD specification) and modifies its definition in order to generate a more well-structured and simplified schema to be further converted. Examples of schema modifications are: removal of elements that are not semantically relevant (for example, an `author-list` element as a component of an element `book`, acting as an intermediate element between `book` and `author` elements); and the replacing of nested components by a new element type (called *virtual element*) that abstracts the set of component elements<sup>2</sup>. Some of these modifications require user intervention, like the first example.

The *Conversion* step takes a pre-processed XML schema and applies a set of conversion rules on it, generating a *preliminary conceptual schema* and mapping information. Section 3.3 presents these rules. The *Restructuring* step takes a preliminary conceptual schema and performs manual and automatic modifications on it to produce a more semantically correct and simplified conceptual schema (a *definitive conceptual schema*). Examples of manual modifications are: definition of suitable names for automatic-generated concepts, and the validation of default cardinality constraints for relationships. An example of automatic modification is the removal of redundant relationships.

The considered XML and conceptual models are presented in the following, for sake of understanding of the conversion rules. It is necessary to introduce again the XML logical model in this paper because BInXS deals with several features of this model that are not fully considered in related work.

#### 3.1 XML Model

The XML logical model defines *elements* and *attributes*. An element is composed by a *start-tag*, a *content model* and an *end-tag*. The content model defines what is enclosed between the *start-tag* and the *end-tag*. An *attribute* describes a property of an element. Its value is specified at the *start-tag* of the element. Figure 2 (a) shows an XML document.  $e_3$  and  $e_{15}$  are elements and  $a_1$  is an attribute of  $e_3$ . Figure 2 (b) shows the correspondent schema to this XML document<sup>3</sup>. A terminology to the concepts of the XML model is presented in the following. Examples are taken from Figure 2(b).

A **composite element** is an element with attributes and/or an element that has a content model defined by one of two XML grammatical constructs: sequence and choice. A *sequence* ( $e_{c_1}, e_{c_2}, \dots, e_{c_n}$ ) defines  $n$  ordered component elements, with  $n \geq 1$ . A *choice* ( $e_{c_1}|e_{c_2}|\dots|e_{c_m}$ ) define  $m$  alternatives for component elements,

<sup>2</sup>  $e_1Group_1$  in Figure 2 (c) is a conceptual abstraction of a virtual element. See sections 3.1 to 3.3.

<sup>3</sup> Figure 2 (b) is a logical abstraction of a schema defined through a DTD (*Document Type Definition*) or an XSD (*XML Schema Definition*) [5].

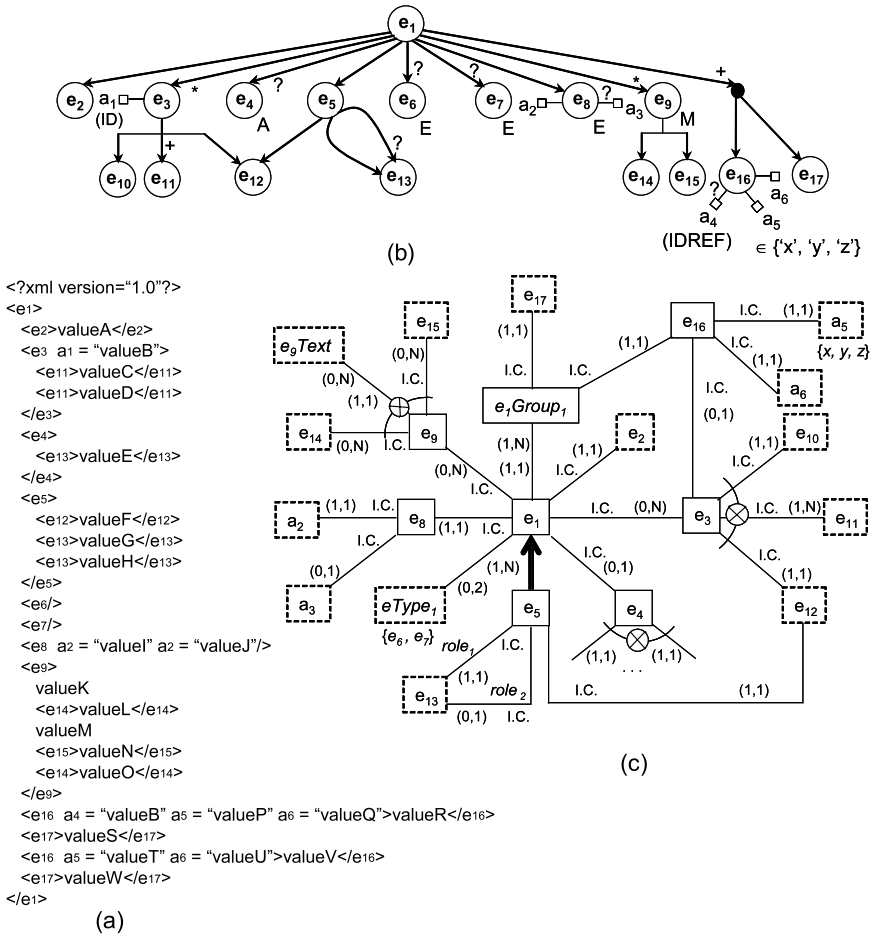


Fig. 2. An XML document (a), an XML schema for the document (b), and a conceptual schema for the XML schema (c)

with  $m > 1$ . The regular expression operators '?', '\*' and '+' indicate the allowed number of occurrences of a composite element, denoting, respectively, 0 to 1 occurrences, 0 to  $n$  occurrences, and 1 to  $n$  occurrences.  $e_1, e_3, e_5, e_8$  and  $e_{16}$  are examples.  $e_3$  is an element defined by a choice and  $e_5$  is an element defined by a sequence. A **nested component** is a *sequence* or *choice* specification that is embedded into the content model of a composite element. The composite element  $e_1$  has a nested component comprised by  $e_{16}$  and  $e_{17}$ .

A **simple element** has a content model defined by a single value.  $e_2, e_{10}$  to  $e_{15}$ , and  $e_{17}$  are examples. An **empty element** has no content model, i.e., its content model is *empty*.  $e_6$  and  $e_7$  are examples (labelled by 'E'). A **free element** allows any kind of schema element in your content model. It corresponds to an ANY element in a DTD or XSD specification.  $e_4$  is an example (labelled by 'A').

A **mixed element** has a content model that is a mix of values and component elements, i.e., it is a **composite element** with the following restrictions: (i) its content model is defined by a choice; (ii) its components may repeat from zero to  $N$  times; (iii) there is a special component (a *valued component*) without a name.  $e_9$  is an example (labelled by 'M'). The elements  $e_{14}$  and  $e_{15}$ , as well as the implicit valued component, may occur from zero to several times into  $e_9$  content.

An **attribute** is an optional or required property associated to an element, like  $a_2$  and  $a_6$ . An attribute has a data type and may act as an element identifier (an ID attribute, like  $a_1$ ). A **reference attribute** for an element  $e_x$  is an **attribute** that establishes a reference from an  $e_x$  element instance to one or more values of ID attributes of instances of elements.  $a_4$  is an example (labelled by 'IDREF' or 'IDREFS').

### 3.2 Conceptual Model

BInXS adopts a graphic variant of the ORM/NIAM (*Object with Roles Model/Natural language Information Analysis Method*) conceptual model as the canonical model [19]. Figure 2 (c) shows an example of a conceptual canonical schema.

The ORM/NIAM model is based on two types of concepts: lexical and non-lexical concepts. A *lexical concept* models information that has an associated value (a dotted rectangle).  $a_5$  and  $e_2$  are examples of lexical concepts. A lexical concept has a data type (*string* or *integer*, for example), and an optional enumeration of allowed values, as shown in the concept  $a_5$ . A *non-lexical concept* models information that is composed by other information (a solid rectangle).  $e_1$  and  $e_8$  are examples of non-lexical concepts. The model supports binary *association relationships* (with optional roles) with cardinality constraints, and *inheritance relationships*. An association relationship is defined between the concepts  $e_1$  and  $e_1Type_1$ , and an inheritance relationship is defined between  $e_1$  and  $e_5$ , being  $e_5$  a specialization of  $e_1$ . It is still possible to model *mutually exclusive relationships*, like the relationships of  $e_3$  with  $e_{10}$ ,  $e_{11}$  and  $e_{12}$ .

The ORM/NIAM model was chosen to be the canonical model because it has a more straight correspondence with the XML logical model: non-lexical concepts are suitable to model composite elements, and lexical concepts are suitable to model simple elements and attributes. Besides, simple elements and attributes (valued information) may be associated to several composite elements in an XML schema. Such situation is also possible in the ORM/NIAM model, i.e., a lexical concept may have relationships with several non-lexical concepts. However, this is not possible in the ER model [9], for example, where valued information can only be modelled as an attribute, which is an exclusive property of an entity or relationship.

### 3.3 Conversion Rules

The *conversion rules* are the core of the *Schema Conversion* phase. They are summarized in the following<sup>4</sup>.

<sup>4</sup> For sake of paper space, correctness and completeness of the conversion rules are not discussed. This is a focus of future work.



**Rule 1 (Simple Element Conversion).** A simple element  $E_S$  generates a *lexical concept*  $E_l$  with name  $E_S$ . The data type of  $E_l$  is the data type defined to the simple element, if exists; or *string*, otherwise.

**Rule 2 (Empty Element Conversion).** An empty element  $E_E$  generates a *lexical concept*  $eType_i$ , where  $i$  corresponds to the  $i$ -esimal converted empty element. The data type of  $eType_i$  is set to *string* and its enumeration is set to  $\{E_E\}$ .

**Rule 3 (Free Element Conversion).** The conversion of a free element  $E_A$  proceeds as follows:

1. a *non-lexical concept*  $E_{nl}$  with name  $E_A$  is generated;
2. given  $n$  the number of lexical or non-lexical concepts  $NL$  that corresponds to XML elements, for  $i$  from 1 to  $n$ : generate an association relationship  $R_i$  between  $E_{nl}$  and  $NL_i$  with a direct cardinality  $(0,1)$  and an inverse cardinality  $(0,N)$ ;
3. all previously defined relationships are set as *mutually exclusive*.

**Rule 4 (Attribute Conversion).** The conversion of an attribute  $a_x$  of a composite element  $E_C$  proceeds as follows:

**IF**  $a_x$  is a **reference attribute** and an analysis of XML documents indicates that all references of  $a_x$  instances points to instances of a same (target) element type  $E_T$

**THEN** generates an association relationship between  $E_C$  and the non-lexical concept corresponding to  $E_T$  with a direct cardinality  $([0-1],1)$ , depending if the attribute is optional or not; and define the inverse cardinality through analysis of XML documents or assume  $(1,N)$  as default

**ELSE** generates a lexical concept  $E_l$  with name  $a_x$ . The data type of  $E_l$  is the data type defined to  $a_x$ , if exists; or *string*, otherwise. If  $a_x$  has an enumeration, it is transferred to  $E_l$ .

**Rule 5 (Composite Element Conversion).** The conversion of a composite element  $E_C$  proceeds as follows:

1. a non-lexical concept  $E_{nl}$  with name  $E_C$  is generated;
2. given  $\{ec_1, ec_2, \dots, ec_n\}$  the set of component elements of  $E_C$ , for each component element  $ec_i$  ( $1 \leq i \leq n$ ):

**IF**  $ec_i$  is not an **empty element** and it is possible to infer an  $\langle E_C \text{ hyperonym } ec_i \rangle^5$  relation with the aid of a lexical database

**THEN** generates an inheritance relationship  $R_i$  between  $E_{nl}$  and the concept correspondent to  $ec_i$

**ELSE** generates an association relationship  $R_i$  between  $E_{nl}$  and the concept correspondent to  $ec_i$  with a direct cardinality based on the defined regular expression operator; and an inverse cardinality defined through analysis of XML documents or assumed as  $(1,N)$  as default;

3. **IF**  $E_C$  is a **mixed element**

**THEN** generates:

- (a) a lexical concept with a name ' $E_C$  +  $\textit{Text}$ ', and a data type *string*;

<sup>5</sup>  $t_1$  hyperonym  $t_2$  means that  $t_1$  is a more general term than  $t_2$ .

- (b) an association relationship between  $E_{nl}$  and  $E_{Text}$  with a direct cardinality  $(0,N)$  and an inverse cardinality  $(1,1)$ ;
4. **IF**  $R_i$  associates  $E_{nl}$  with a lexical concept  $L_a$  generated from an **empty element** and there is another lexical concept  $L_b$  also generated from an empty element and an association relationship  $R_j$  between  $E_{nl}$  and  $L_b$  with the same direct cardinality **THEN** merges  $L_a$  and  $L_b$  into a lexical concept  $L_u$ , and merges  $R_i$  and  $R_j$  into an association relationship  $R_u$  between  $E_{nl}$  and  $L_u$ , adjusting properly the direct cardinality. The set of enumerations of  $L_a$  and  $L_b$  are also unified;
  5. **IF** there is more than one relationship  $R_1, R_2, \dots, R_k$  between  $E_{nl}$  and a concept  $C_x$  **THEN** defines default names  $role_1, role_2, \dots, role_k$  to each respective relationship;
  6. given  $\{a_1, a_2, \dots, a_m\}$  the set of attributes of  $E_C$ , for each attribute  $a_i$  ( $1 \leq i \leq m$ ) generates an association relationship between  $E_{nl}$  and  $a_i$  with a direct cardinality  $(1,1)$  or  $(0,1)$ , depending if  $a_i$  is required or not, respectively; and an inverse cardinality defined through analysis of XML documents or assumed as  $(1,N)$  as default;
  7. **IF** the content model of  $E_C$  is defined by a *choice* **THEN** set all previously defined relationships as *mutually exclusive*.

Figure 2 (c) is the preliminary conceptual schema generated by the application of the conversion rules on the XML schema in Figure 2 (b)<sup>6</sup>. The concepts  $e_2$  and  $a_2$ , for example, are created by **Rule 1** and **Rule 4** applied to the element  $e_2$  and the attribute  $a_2$ , respectively. **Rule 4** is also applied to the reference attribute  $a_4$ , defining an association relationship between the concepts  $e_3$  and  $e_{16}$ . **Rule 3** applied to the element  $e_4$  generates a same name concept and their mutual exclusive relationships with other element-derived concepts. **Rule 2** applied to the elements  $e_6$  and  $e_7$  generates the concepts  $eType_1$  and  $eType_2$ , that are further merged into a single concept  $eType_1$  by the application of **Rule 5** to the element  $e_1$ . It means that an empty element of a composite element  $E_C$  is considered a property (or *qualification*) of  $E_C$ , being represented as a lexical concept associated to it with a fixed value. Empty elements with the same direct cardinality are merged into a single lexical concept, with a set of allowed values.

Besides generating a concept  $e_9$ , **Rule 5** applied to the mixed element  $e_9$  generates a new concept  $e_9Text$  that abstracts its valued components, and a set of mutually exclusive relationships that comprises the relationship to  $e_9Text$  and all relationships to the concepts generated to its component elements, considering that the content model of  $e_9$  is defined by a choice. **Rule 5** applied to the element  $e_5$  generates two association relationships from the  $e_5$  concept to the  $e_{13}$  concept. Because of this, two default role names are defined to these relationships. In the *Restructuring* step, these names may be changed by the user, or the relationships may be merged if the user assumes that they have the same semantic meaning.

### 3.4 Mapping Strategy

Mapping information are defined during the *Conversion* step to each generated concept or relationship in the conceptual schema. BInXS adopts *XPath 1.0 expressions* to specify mappings to an XML schema [6]. *XPath* was chosen because it is a W3C recommendation for searching elements and attributes in an XML document.

<sup>6</sup> 'I.C.' denotes an automatic-generated default cardinality.

The mapping of a concept  $C_g$  is defined as an *absolute path expression* in *XPath*, i.e., a complete path from the root element to the  $C_g$  correspondent element or attribute in the XML schema. Given the conceptual and XML schemata in Figure 2 (c) and Figure 2 (b), the mapping of the concepts  $e_{14}$  and  $a_2$  are denoted respectively by the expressions  $'/e_9/e_{14}'$  and  $'/e_8/@a_2'$ .

The mapping of a relationship is defined as a *relative path expression* in *XPath*. Such expression says how to navigate between related concepts in an XML schema. Mappings are defined for both relationship directions in order to allow the translation of any traversal over the conceptual schema graph. In Figure 2 (c), the *XPath* expressions  $'e_{14}'$  and  $'..'$  denote, respectively, the mapping of the relationship between the concepts  $e_9$  and  $e_{14}$  in the directions  $e_9 \rightarrow e_{14}$  and  $e_9 \leftarrow e_{14}$ .

A query language for conceptual schemata called *CXPath* (*Conceptual XPath*) was defined in the context of the BInXS approach. A *CXPath* query is an *XPath*-like query that starts at a concept and traverses the schema graph in any direction in order to reach a desired related concept. With the proposed mapping strategy, the translation of a *CXPath* query does not become complex because the translation process will basically replace the concepts as well as the relationship traversals in a *CXPath* expression by their correspondent mappings in *XPath* to the schema of an XML source  $XS_i$ . Once unified, these *XPath* expressions define a complete *XPath* query to be executed at  $XS_i$ <sup>7</sup>.

## 4 Unification

Once defined a set of conceptual schemata from local schemata<sup>8</sup>, the *Unification* phase performs their semantic integration, generating a *global schema* [26]. To each global concept or relationship are associated the mappings to all respective local concepts or relationships that it represents. These mappings are kept in a global catalog. This phase follows the traditional database schema integration steps: *Schema Comparison*, *Merging* and *Restructuring* [10, 17].

The *Schema Comparison* step defines groups of synonym concepts coming from different local schemata called *affinity clusters*. An affinity cluster belongs to one of the following types: a *lexical cluster*, that holds only lexical concepts; a *non-lexical cluster*, that holds only non-lexical concepts; and a *mixed cluster*, that holds lexical and non-lexical concepts. The definition of these clusters is supported by an external tool called ARTEMIS [14], and it is out of the scope of this paper.

The *Merging* step is the core of the *Unification* phase. It generates concepts and relationships of a *preliminary global schema* through the merging of concepts in a same affinity cluster. Such merging is based on semi-automatic unification rules that are applied on the context of three unification cases: **LxL** (*lexical unification*), **NLxNL** (*non-lexical unification*), and **NLxL** (*mixed unification*). The next sections detail these cases.

Once performed the *Merging* step, the resulting *preliminary global schema* is validated in the *Restructuring* step through a set of automatic, semi-automatic and manual

<sup>7</sup> For sake of paper space, *CXPath* and the translation process are not detailed. See [13].

<sup>8</sup> From now on, XML schemata are called local schemata.

actions to generate the *definitive global schema*. An example of semi-automatic action is the definition of new inheritance relationships between global concepts with some common properties coming from different local schemata. Such relationships are defined with the aid of terminological databases and further user validation. An automatic action is the generalization of association relationships defined to all specialized concepts in an inheritance hierarchy. Manual adjustments include names of new concepts and relationship cardinalities.

#### 4.1 Lexical Unification

The **LxL case** merges the concepts of a lexical cluster, generating a lexical concept  $L_G$  at the global schema. It corresponds to the merging of all XML valued content with affinity in different local schemata: simple elements, empty elements (considered properties), attributes and valued components of mixed elements. Specific rules determine the name, data type and allowed values of  $L_G$ .

Figure 3 shows the unification of two local schemata:  $S_1$  and  $S_2$ . This example is used to illustrate all unification cases. Several lexical clusters (denoted by ( $L$ )) are defined between local concepts, like 1 and 2. Cluster 1 generates the global concept *Style*, with a name chosen by the user between the names in the cluster. As both of the local concepts have enumerations, they are also unified. Cluster 2 generates the global concept *University*, whose name is the one with more incidences in the cluster.

#### 4.2 Non-lexical Unification

The **NLxNL case** merges the concepts of a non-lexical cluster, generating a non-lexical concept  $NL_G$  at the global schema. It corresponds to the merging of all XML element types that are composed by other elements or attributes: composite elements, mixed elements and free elements.

To merge relationships, an iterative matching of pairs of concepts in the cluster is provided, until one single concept ( $NL_G$ ) exists in the cluster. Basically, at each iteration it is analyzed if two relationships have *affinity*. Consider two concepts  $c_i$  and  $c_j$  in the same cluster. A relationship  $r_i$  of  $c_i$  has affinity with a relationship  $r_j$  of  $c_j$  if: (i)  $r_i$  and  $r_j$  have the same type (association or inheritance) and; (ii) both of them associate  $c_i$  and  $c_j$  with concepts in the same affinity cluster  $AC_L$ . If so, a merged relationship  $r_{ij}$  is generated from  $NL_G$  to the concept that represents  $AC_L$  in the global schema. User intervention is required when an association relationship  $r_i$  has affinity with more than one relationship of  $c_j$  (or vice-versa). Such situation raises when  $c_j$  has two or more association relationships with a same concept, and these relationships have roles. In this case, the user must decide if  $r_i$  has affinity with one of the  $c_j$  relationship or not. If a  $c_i$  or  $c_j$  relationship has no affinity with other relationships, it is considered an optional  $NL_G$  relationship.

The affinity cluster 13 is an example of non-lexical cluster, that generates the global concept *Address*. The relationships *Address-Country* ( $S_1$ ), *Address-Street* ( $S_2$ ) and *Address-ZipCode* ( $S_2$ ) become optional relationships because they have no affinity with other relationships. The relationships *Address-City* in  $S_1$  and  $S_2$  have affinity and are unified. The cardinality constraints are adjusted to be in accordance to both local cardinality constraints. The relationship *Address-Author* in  $S_2$  has affinity with two  $S_1$

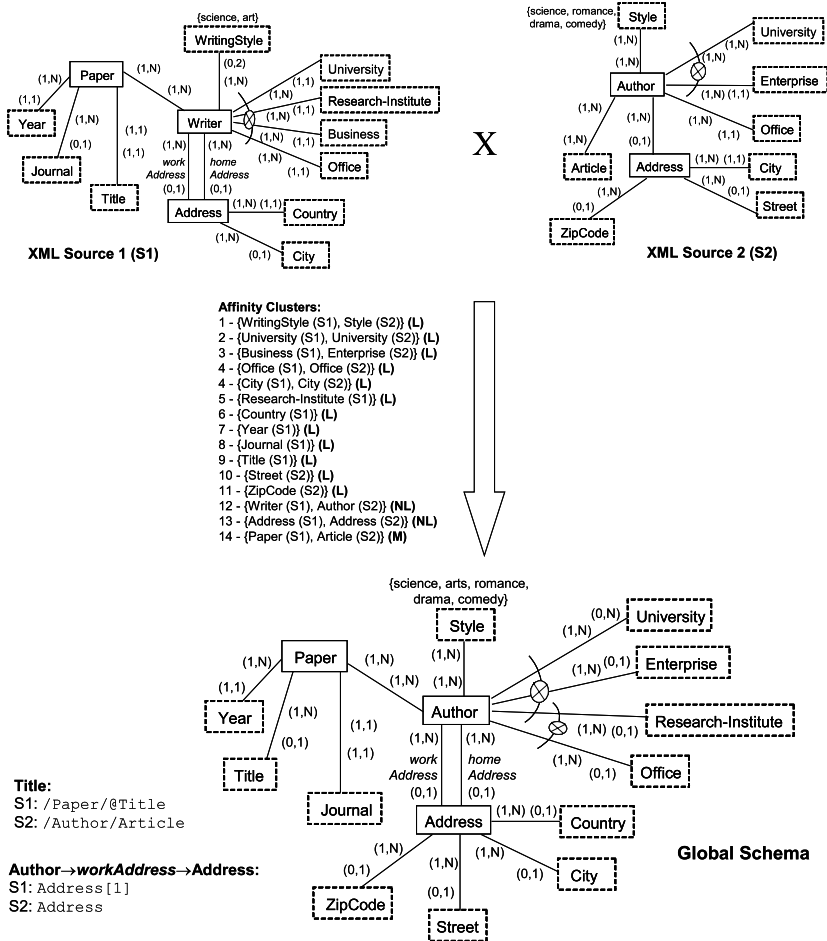


Fig. 3. An example of unification of two local schemata

relationships: (i) *Address-homeAddress-Writer* and (ii) *Address-workAddress-Writer*. Supposing that user intervention had decided by an affinity with relationship (ii), it is indicated in the global schema that the relationship *Address-workAddress-Author* has mappings to S1 and S2.

A mutual exclusion constraint defined to  $c_i$  relationships, for example, is directly represented at  $NL_G$  if such relationships have no affinity with  $c_j$  relationships. Otherwise, it is possible that a mutual exclusion constraint conflict exists, and a detailed analysis of  $c_i$  and  $c_j$  relationships must be performed<sup>9</sup>. In this case, only *valid mutual exclusions* are considered over  $NL_G$  relationships. Basically, a *valid mutual exclusion*

<sup>9</sup> This conflict is implicitly related to the problem of unifying XML elements with alternative representations.

is the one that comprises: (i)  $c_i$  disjoint relationships  $ri_1, \dots, ri_n$ , and  $c_j$  disjoint relationships  $rj_1, \dots, rj_n$  with affinity and; (ii) other  $c_i$  and  $c_j$  relationships that have no affinity but are disjoint of  $ri_1, \dots, ri_n$  and  $rj_1, \dots, rj_n$ , respectively. A subset of  $c_i$  and  $c_j$  relationships in a local mutual exclusion constraint without relationship affinity, or at most with one relationship with affinity, is also a valid mutual exclusion (case (iii)). Cases (ii) and (iii) preserve local mutual exclusion constraints at the global level.

The unification of the affinity cluster 12 in Figure 3 raises a mutual exclusion constraint conflict among the relationships of the local concepts *Writer* and *Author*. The conflict resolution performs as follows: the relationships with *University* and *Enterprise* are mutually exclusive in both local schemata. Therefore, an exclusion constraint  $me_i$  is defined between them in the global schema (case (i)). The relationship *Writer-Research-Institute* in  $S_1$  has no affinity with  $S_2$  relationships but is mutually exclusive of the two concepts mentioned above. Therefore, it is included in  $me_i$  in order to maintain the  $S_1$  constraint (case (ii)). Besides, the relationship subset that comprises *Writer-Research-Institute* and *Writer-Office* is still mutually exclusive in  $S_1$ . As *Writer-Office* is the only relationship with affinity, an exclusion constraint is defined on them at the global schema (case (iii)).

Again, observe that the global relationships *Author-University*, *Author-Enterprise* and *Author-Office* are defined as optional *Author* relationships. Such definitions avoid that, for example, *Author-University* and *Author-Office* always occur simultaneously at the global level, considering that their correspondent relationships in  $S_1$  are mutually exclusive. Such analysis is also performed during the resolution of mutual exclusion conflicts.

### 4.3 Mixed Unification

The **NLxL case** merges all the concepts of a mixed cluster, generating a global non-lexical concept  $NL_G$ . It corresponds to the merging of structured and valued information with affinity in different local schemata.

The unification proceeds as follows: first, all non-lexical concepts are unified into a preliminary non-lexical concept  $NL_P$  by the application of the **NLxNL case**. After, for each remaining lexical concept  $L_i$  in the cluster, the user decides by one of the following alternatives: (i)  $L_i$  is mapped to a global lexical concept related to  $NL_P$ , assuming that  $L_i$  has a semantic correspondence with a  $NL_P$  property; (ii)  $L_i$  becomes a global concept and a new non-lexical concept  $NL_U$  is defined as a mutually exclusive generalization of  $L_i$  and  $NL_P$ . Such alternative assumes that  $L_i$  corresponds to the union of two or more  $NL_P$  properties, and must be denoted as an alternative representation for  $NL_P$  at the global level; (iii)  $L_i$  becomes a global concept associated to  $NL_P$ , assuming that  $L_i$  has no semantic correspondence with  $NL_P$  properties.

The affinity cluster 14 in Figure 3 is an example of a mixed cluster composed by the lexical concept *Article* and the non-lexical concept *Paper*. Considering that *Article* keeps titles of articles in  $S_2$ , it corresponds to the lexical concept *Title* associated to *Paper* in the global schema (alternative (i)). Then, this mapping to the concept *Title* is also kept in the global schema, as shown in Figure 3. Alternative (ii) could be applied if, for example, *Article* content was a complete bibliographic reference, including not only a title, but also other reference information. In this case, a non-lexical concept

*GenericPaper* could be defined as a mutually exclusive generalization of *Paper* and *Article*, representing an abstraction of two possible disjoint representations for a *paper*.

## 5 Related Work

There are several work related to the integration of semistructured data [11, 21, 23, 24] or XML sources [15, 16, 20, 22, 28, 29, 30, 32]. Some of them gives support to a manual integration process, acting only as a tool that aids the user to define global views or mappings among local schemata [24, 30]. Thus, their integration process has a low quality because they provide a weak automation level. Another point is the canonical model. There are work that deal with hierarchical models for semistructured data as the canonical model, or performs the integration straightly over the XML model [15, 16, 24, 29, 32]. As a conceptual schema is not considered, their models enforce the structural organization of data instead of data semantics.

An approach different from BInXS is followed by [24], that defines mappings among local schemata instead of creating a global schema. This alternative is not adopted by BInXS because we are considering the context of the Web, where there are a lot of available XML sources. In this context, it is preferable to define a global representation of these sources in order to provide an integrated access to them.

Close related work are [11, 20, 21, 23, 28], which also propose semi-automatic schema integration of conceptual representation of semistructured schemata. However, they do not consider all features of the XML logical model, like elements with alternative representations, mixed elements and references between elements, or do not detail the mixed unification case as BInXS does. In [22], it is proposed an ER-like conceptual model for representing XML data that considers XML hierarchical relationships between elements in the conceptual schemata. As the same related real world facts may be expressed by different hierarchies in two or more XML schemata, this model is not suitable to represent an integrated view of these schemata.

## 6 Conclusion

BInXS is a solution to the problem of schema integration for XML data. The focus on XML schemata is justified by the widespread use of XML protocols by users and applications to represent and interchange data, specially over the Web. The bottom-up approach followed by BInXS is suitable to the context of the Web because it provides an unified view of a lot of heterogeneous XML sources over the Web. If used as a basis for querying XML data sources, this unified view avoids that users and applications must know the schema of each XML source in order to formulate a query.

Compared to related work, the main contributions of BInXS are the following:

- *A semi-automatic conversion process of an XML schema to a conceptual schema:* this process is based on a detailed analysis of the XML logical model and XML documents in order to obtain a correspondent conceptual abstraction where data semantics is much clear. The proposed conceptual representation is able to model all

types of elements (simple, composite, mixed, etc); attributes; element-to-element association, element-to-attribute association, references between elements; inferred inheritance relationships between elements; and alternative representations for elements;

- *A semi-automatic unification process for conceptual representations of XML schemata*: this process is suitable to XML schema integration because takes into consideration the implicit merging of heterogeneous XML data, with content models that may hold a value, a structure composed by other XML data, a mix of value and structure, and have alternative representations;
- *A mapping strategy between a global schema and an XML schema*: the *XPath* language is used to define mapping expressions from conceptual data to XML data. Because *XPath* is a language for querying XML data, a query defined over the global schema is easily translated to an *XPath* query to be executed at an XML source. No similar strategy was found in related work.

As user expertise is considered in the process, a good integration quality is always expected. However, future work include the consideration of instance-based integration techniques at BInXS with the purpose of improving the quality of the results generated automatically. On combining schema and instance analysis of XML sources, it is possible to establish semantic correspondences with much precision. The consideration of semantic integrity constraints of local XML sources is also important. Such information could be available and associated to concepts and relationships of the global schema in the global catalog. Thus, if a global query  $q_i$  defines a selection predicate that is not in accordance to the semantic constraints of an XML source  $XS_i$ ,  $q_i$  does not need to be translated to  $XS_i$  because no XML instances will be retrieved from there.

## References

1. CXML.org. Available at: <http://www.cxml.org>, mar 2005.
2. DBLP Bibliography. Available at: <http://www.informatik.uni-trier.de/~ley/db/>, mar 2005.
3. EBisXML. Available at: <http://www.basda.org>, mar 2005.
4. SIGMOD Record. Available at: <http://www.acm.org/sigs/sigmod/record/xml>, mar 2005.
5. W3C XML Schema. Available at: <http://www.w3.org/XML/Schema>, mar 2005.
6. XML Path Language. Available at: <http://www.w3.org/TR/xpath>, mar 2005.
7. Extensible Markup Language (XML). Available at: <http://www.w3.org/XML>, mar 2005.
8. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, San Francisco, California, 2000.
9. C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin/Cummings Publishing Company, 1992.
10. C Batini, M. Lanzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, december 1986.
11. S. Bergamaschi, S. Castano, D. Beneventano, and M. Vincini. Semantic Integration of Heterogeneous Information Sources. *Data & Knowledge Engineering*, 36(1):215–249, march 2001.
12. S. Busse, R. Kutshce, U. Leser, and H. Weber. *Federated Information Systems: Concepts, Terminology and Architectures*. (Technical Report, 99-9), Berlin: Universitt Berlin, 1999.



13. S. D. Camillo, C. A. Heuser, and R. S. Mello. Querying Heterogeneous XML Sources Through a Conceptual Schema. In *22th International Conference On Conceptual Modeling (ER)*, pages 186–199, Chicago, USA, 2003. Springer-Verlag.
14. S. Castano, V. Antonellis, and S. C. Vimercati. Global Viewing of Heterogeneous Data Sources. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):277–297, march 2001.
15. I. F. Cruz, H. Xiao, and F. Hsu. An Ontology-Based Framework for XML Semantic Integration. In *International Database Engineering and Applications Symposium (IDEAS'04)*, pages 217–226, Coimbra, Portugal, 2004. IEEE.
16. A. Doan, P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In *ACM International Conference on Management of Data (SIGMOD)*, pages 509–520, Santa Barbara, USA, may 2001.
17. A. Elmagarmid, M. Rusinkiewicz, and A. Sheth. *Management of Heterogeneous and Autonomous Database Systems*. Morgan Kaufmann, San Francisco, California, 1999.
18. A. Y. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, 10(4):270–294, december 2001.
19. T. Halphin. *Object-Role Modeling (ORM/NIAM), Handbook on Architectures of Information Systems*, chapter 4, pages 81–102. Springer-Verlag, 1998.
20. P. Lethi and P. Fankhaue. XML Data Integration with OWL: Experiences & Challenges. In *International Symposium On Applications and the Internet (SAINT'2004)*, pages 160–170, Tokyo, Japan, 2004. IEEE.
21. S. Lim and Y. Ng. An Automated Integration Approach for Semi-structured and Structured Data. In *3th International Symposium on Cooperative Database Systems for Advanced Applications (CODAS)*, pages 12–21, Beijing, China, april 2001. IEEE.
22. B. F. Lscio and A. C. Salgado. Generating Mediation Queries for XML-based Data Integration Systems. In *18th Brazilian Symposium on Databases (SBBD'2003)*, pages 99–113, Manaus, AM, october 2003.
23. J. Madhavan, P. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *27th Conference on Very Large Data Bases (VLDB)*, pages 49–58, Rome, Italy, september 2001. Morgan Kaufmann.
24. P. McBrien and A. Poulouvasilis. A Semantic Approach to Integrating XML and Structured Data Sources. In *13th Conference On Advanced Information Systems Engineering (CAISE)*, pages 330–345, Interlaken, Switzerland, 2001. Springer-Verlag.
25. R. S. Mello. *Uma Abordagem Bottom-Up para a Integracao Semantica de Esquemas XML*. PhD thesis, Universidade Federal do Rio Grande do Sul, july 2002. (In Portuguese).
26. R. S. Mello, S. Castano, and C. A. Heuser. A Method for The Unification of XML Schemata. *Information and Software Technology*, 44(4):241–249, march 2002.
27. R. S. Mello and C. A. Heuser. A Rule-Based Conversion of a DTD to a Conceptual Schema. In *20th International Conference On Conceptual Modeling (ER)*, pages 133–148, Yokohama, Japan, 2001. Springer-Verlag.
28. K. Passi, L. Lane, S. K. Madria, B. C. Sakamuri, M. K. Mohania, and S. S. Bhowmick. A Model for XML Schema Integration. In *3th International Conference On E-Commerce and Web Technologies (EC-WEB 2002)*, pages 193–202, France, 2002. Springer-Verlag.
29. C. Reynaud, J. Sirot, and D. Vodislav. Semantic Integration of XML Heterogeneous Data Sources. In *International Database Engineering & Applications Symposium (IDEAS)*, pages 199–208, Grenoble, France, july 2001. IEEE.
30. P. Rodriguez-Gianolli and J. Mylopoulos. A Semantic Approach to XML-Based Data Integration. In *20th International Conference On Conceptual Modeling (ER)*, pages 117–132, Yokohama, Japan, 2001. Springer-Verlag.

31. A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, september 1990.
32. X. Yang, M. L. Lee, and T. W. Ling. Resolving Structural Conflicts in the Integration of XML Schemas: A Semantic Approach. In *22th International Conference On Conceptual Modeling (ER)*, pages 520–533, Chicago, USA, 2003. Springer-Verlag.

# Query Processing Using Ontologies

Chokri Ben Necib and Johann-Christoph Freytag

Humboldt-Universität zu Berlin, Germany

{necib, freytag}@dbis.informatik.hu-berlin.de

**Abstract.** Recently, the database and AI research communities have paid increased attention to *ontologies*. The main motivating reason is that ontologies promise solutions for complex problems caused by the lack of a good understanding of the semantics of data in many cases. In particular, ontologies have extensively been used to overcome the interoperability problem during the integration of heterogeneous information sources. Moreover, many efforts have been put into developing ontology based techniques for improving the query answering process in database and information systems.

In this paper, we present a new approach for query processing within single (object) relational databases using ontology knowledge. Our goal is to process database queries in a semantically more meaningful way. In fact, our approach shows how an ontology can be effectively exploited to rewrite a user query into another one such that the new query provides more meaningful results satisfying the intention of the user. To this end, we develop a set of transformation rules which rely on semantic information extracted from the ontology associated with the database. In addition, we propose a semantic model and a set of criteria to prove the validity of the transformation results. We also address the necessary mappings between an ontology and its underlying database w.r.t. our framework.

## 1 Introduction

With the rapid growth of data in databases and information sources and the increasing demands for exchanging information through the internet, the challenges in accessing data become more complex than in past few decades. The major problems are: (i) hiding the heterogeneity in format and structure of data from the users, (ii) overcoming the confusion in terminologies caused by employing synonyms and homonyms, and (iii) providing users with the most relevant answers to his requests in less time and/or resources. Therefore, the need to "understand" data of the information sources is increasing. Web search engines, for example, try to replace their syntactic based retrieval of information by a semantic based one [5]. In this context, researchers become aware of the usefulness of semantic knowledge to deal with the problems above. Indeed, semantic knowledge about a specific source can be considered as a meta-data layer over the instances of the underlying source.

Recently, *ontologies* have become popular candidates to capture such semantics. The reason is that an ontology can provide a shared common understanding of the application domain in concise and consensual manners. In fact, ontologies provide the

meaning of terms and their relationships by which the domain is modelled [20]. They have been proven to be an important support for managing data in database and information systems for overcoming the interoperability problem of heterogeneous information sources. Thus, users should not care about where and how the data are organized in the sources. For this reason, in systems like OBSERVER [12] and TAMBIS [16] users formulate their queries over a given ontology without directly accessing the data sources themselves. In the meantime, ontologies are also used to enhance the functionality of Web search engines by associating meaning with the content of Web pages. Several approaches propose to annotate Web resources with ontology knowledge and inference mechanisms to improve the search [19, 9]. These efforts, among others, converge to build the so called *Semantic Web*.

In this paper, we present a new approach on how to improve the answers of database queries based on semantic knowledge expressed in ontologies. Given a database, we assume the existence of an ontology which is associated with the database and which provides the context of its objects. We show how an ontology can be exploited effectively to reformulate a user query such that the new query can provide more "meaningful" answer meeting the intention of the user. A query can be defined by a set of selections and projections over database objects satisfying a set of conditions. These conditions are defined by a set of terms and determine the answer to the query. If a user wants to retrieve information from a database about certain objects, he might use terms, which do not exactly match the database values (due to the mismatch between the user's world view and the database designer's world view). However, there might be values in the database that are syntactically different from one another but semantically equivalent to the user terms and that express the same intention of the user. We address this issue as a semantic problem rather than as a pattern matching problem.

The remainder of this paper is organized as follows. First, we state the problem illustrating it by some examples. Then, the concepts of "ontologies" are described. In Section 4 we present our approach for query processing and propose necessary reformulation rules. In order to prove the soundness of the approach a semantic model and a set of criteria are proposed in Section 5. Mappings used to connect an ontology to its underlying database are discussed in Section 6. Finally, Section 7 concludes the paper.

## 2 Motivation and Problem Statement

Traditional techniques for query processing which rely on syntactic approaches become insufficient to cope with problems caused by the heterogeneity of data in its format and structure [22]. Current database and information systems require "more knowledge" about information sources in order to retrieve data in an efficient manner and satisfy the user expectations. For instance, semantic knowledge in the form of integrity constraints have been extensively used for developing query optimization techniques. There, the goal is to rewrite a user query into another query which can return the same result in less time and/or less resources [3]. This paper outlines a new approach for query processing which exploits data semantics in forms of ontologies to provide users with

**Table 1.** Item relation

A-ID	Name	Model	Price
123	computer	ibm	3000 \$
124	intelPc	toshiba	5000 \$
125	notebook	dell	4000 \$
127	pc	compaq	2500 \$
128	product	hp	3000 \$
129	monitor	elsa	1000 \$
135	keyboard	itt	80 \$
136	desktop	ibm	1000 \$
140	macPc	mac	2000 \$
141	calculator	siemens	1500 \$

**Table 2.** Component relation

S-ID	M-ID
123	129
123	135
123	136
124	129
124	135
124	136
125	135
127	129
127	135
127	136
128	129
128	135
128	136
140	129
140	135
140	136
141	135

meaningful answers to their queries. The basic idea is to allow the DBMS to deal with user queries both at the semantic as well as the syntactic level. There, users do not need to fully understand the database content to issue their queries and the resulting database answers could fulfil completely their expectations. In fact, if a user attempts to retrieve information about certain objects from a database, the answer to his query might not satisfy his needs. This can be justified by several facts. First, the information stored in databases are usually captured in natural languages. This leads to several variations in expression of the same concept (synonym problem). Moreover, languages introduce multiple meanings of the same expression (homonym problem). These problems might affect the query results when formulating queries using certain terms. Second, there might be different ways to formulate a query using semantically equivalent terms. We define two sets of terms to be semantically equivalent if they have the same meaning i.e. if their relevant concepts and relationships in the ontology identify the same concept. For example, if two terms are synonyms, they are semantically equivalent. There might be several such sets. Therefore, when a user formulates his query, he might use terms partially covering these semantics. Third, some results in the answer might not be related to the same context associated with the query. The context must be defined by the user. We consider the following example to illustrate these ideas throughout the paper.

**Example 1:** Assuming we have a relational database, denoted by  $DB_1$ . This database contains information about technical items of a store and includes two relations called 'Item' and 'Component': The relation Item contains a set of items described by the attributes 'name', 'model' and 'price'. The relation component contains the parts belonging to each item. The relational schema of  $DB_1$  is described as follows:

**ITEM(A-ID, Name, Model, Price)**

A-ID: Item identifier

Name: Name of the Item

Model: Model of the Item

Price: Price of the Item

PrimaryKey(A-ID)

**COMPONENT (S-ID, M-ID)**

M-ID: Main part identifier

S-ID: Second part identifier

Foreign-Key(M-ID) TO ITEM

Foreign-Key(S-ID) TO ITEM

Primary-Key(S-ID,M-ID)

Suppose, at present, that  $DB_1$  contains the instances as shown in the Tables 1 and 2. Querying the database  $DB_1$  to retrieve information about the Item "computer" also means information about the Items "data processor" and "calculator" because these terms are synonymous with the term "computer". Consequently, if a user formulates his query specifying only the term "computer" he might miss other tuples concerning "data processor" and "calculator". In addition "computer" is implied by other terms which should be considered in the query. This example seems to be simple, but there could be more complicated ones depending on the nature of the query as we shall see later. In fact, the difference between the user's perception of real world objects and the database designer, who registers information about these objects, might cause semantic ambiguities including a "vocabulary problem". Therefore, it is hard for the DBMS to solve such semantic problems without additional semantic knowledge like ontologies.

In summary, we state our problem as follows:

Given a database  $DB$ , an ontology  $O$  and a user query  $Q$ , find a reformulated query  $Q'$  of  $Q$  by using  $O$  such that  $Q'$  returns more meaningful answer to the user than  $Q$ .

### 3 Ontology

#### 3.1 Definition

In recent years, the term "Ontology" has become a "buzz word" for researches in the fields of databases and artificial intelligence. There are many definitions of what an ontology is [7, 8, 15, 4, 19]. An initial definition was given by Tom Gruber: "*An ontology is an explicit specification of a conceptualization*" [7]. Ontologies have been increasingly emerging because of the crucial role that they play: Ontologies provide a concise and unambiguous description of concepts and their relationships for a domain of interest. This knowledge can be shared and reused by different participants.

Informally, we define an ontology as an intentional description of what is known about the essence of the entities in a particular domain of interest using abstractions, also called *concepts* and the *relationships* among them. Basically, the hierarchical organization of concepts through the inheritance ("ISA") relationship constitutes the backbone of an ontology. Other kinds of relationship like part-whole ("PartOf") or Synonym ("SynOf") or application specific relationships might also exist. Furthermore, a set of logical axioms is often associated with the ontology to specify semantics of the relationships. To the best of our knowledge, there is no work until now addressing the issue of using ontology relationships at the database instance level.

For clarity, we have to distinguish between the meaning of the term "concept" and that of the term "concept instance". A concept is a description of a group of real world

objects in a certain domain whereas a concept instance is a set of values that represent these objects [2]. Note that many real-world ontologies already combine data instances and concepts [8]. In our definition we do not consider instances as part of an ontology.

For the remainder of the paper we refer to the set of the ontology concepts as  $\zeta = \{c_1, \dots, c_n\}$  and the set of ontological relationships as  $\mathfrak{R} = \{ \text{"ISA"}, \text{"SynOf"}, \text{"PartOf"}, \dots \}$ , where  $c_i \in \zeta$  and  $r_i \in \mathfrak{R}$  are non-null strings. We denote the set of axioms by  $\mathfrak{S}$ .

### 3.2 Graphical Representation

An ontology can be then represented as a directed labelled graph  $G(V, E)$ , where  $V$  is a finite set of vertices and  $E$  is a finite set of edges: Each vertex of  $V$  is labelled by a concept from  $\zeta$  and each edge of  $E$  is labelled by an inter-concept relationship from  $\mathfrak{R}$ . Note that instances are not represented in  $G$  because they do not belong to an ontology. Further, we refer to a node by its label (a concept) and refer to an edge by its node concepts and its label (a relationship). For instance, the statement  $e = c_1 R_i c_2$  refers to the edge between the concept nodes  $c_1$  and  $c_2$  which is labelled by a relationship  $R_i$ . Formally, the graph  $G$  can be expressed as a relation  $G \subseteq \zeta \times \mathfrak{R} \times \zeta$ . Appendix A gives the most important graph operations that are used to extract concepts for query reformulations.

Figure 1 gives an example of a graph representation of a fragment of an ontology called "Product Ontology" (denoted by  $O_1$ ). The ontology describes concepts and their

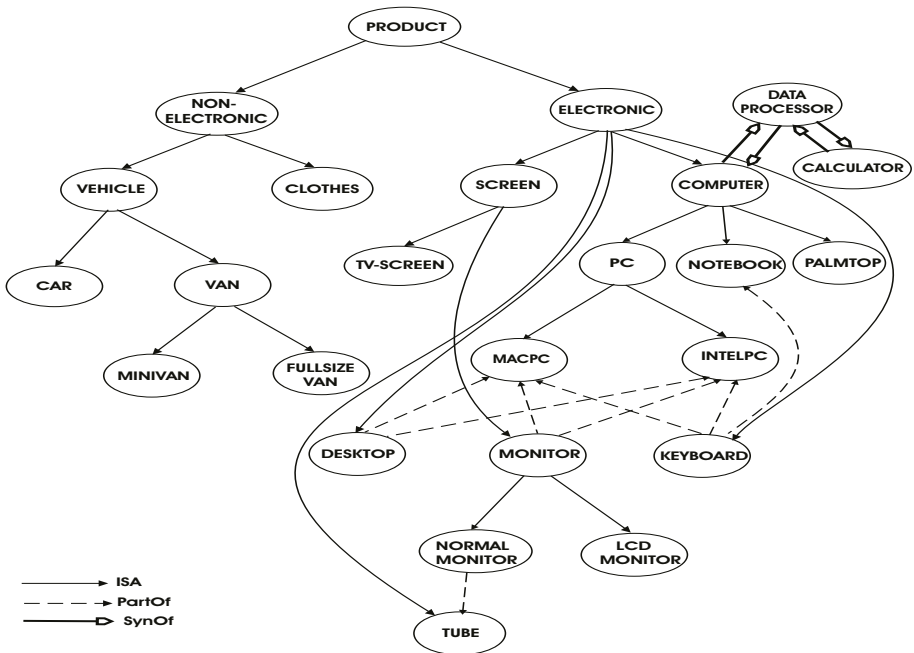


Fig. 1. Product Ontology  $O_1$

relationships related to products. A part of this ontology is adopted from an ontology described in [11].

In summary, we define an ontology as the following set:  $O = \{G, \zeta, \mathcal{R}, \mathcal{S}\}$ .

### 4 Ontology Based Query Processing

The objective of our approach to query processing is to determine an alternative way to reformulate an input query into another "meaningful" query but not necessary equivalent one. The approach can be applied to the DBMS in a simple manner without any complex modifications of its core. Figure 2 shows an overview on the system's architecture. The system mainly consists of three components. The first component is the transformation engine which constitutes the core of the system. It performs a pre-processing of an input query, say  $Q$ , before submitting it to the database. This is done by reformulating  $Q$  into another query, say  $Q'$ , in a semantic meaningful way using a set of semantic rules. These rules rely on additional semantics extracted from an ontology. Basically, they must contribute to:

- Expand user queries by changing their select conditions using synonyms for the terms in the condition and others specifying them.
- Substitute the query conditions with other conditions that are semantically equivalent.
- Reduce the scope of queries by restricting its context (see section.

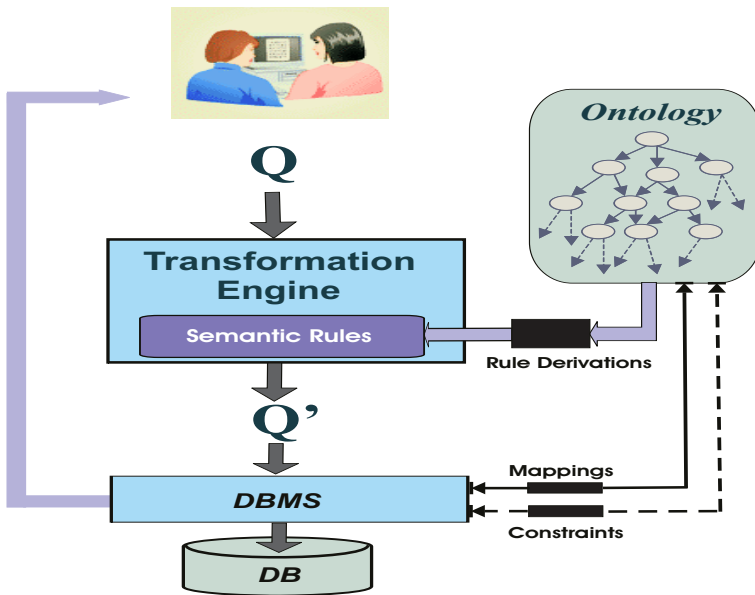


Fig. 2. System Architecture



During query reformulation semantic rules are applied uniformly, in any order. This is done iteratively such that at each iteration the reformulated query obtained in the previous iteration is used to generate another query until no more reformulations are possible. It is possible that no rules can be applied to the query and the output query is then equal to the input query. The rule derivation process is done manually by ontology and database experts. We have developed a set of such rules based on information mappings between ontological and database entities. The second component is an ontology which is associated with the underlying database. It could be either a general or a domain-oriented ontology depending on the nature of the database in question. Here, the role of the ontology is to provide semantic knowledge about the data in the database. Its content is adapted to the database instances in such way that it should be used correctly and completely (see section 5). The dashed arrow represents a set of constraints that must be satisfied for this purpose. The third component is the DBMS which processes the output query and returns the answer to the user. The answer might contain more or fewer tuples than that answer expected by the original query. According to this feature we classify the reformulation rules into two categories: *Augmentation rules* and *reduction rules*. In this paper we focus on the second class. For the first class we describe only one rule; please refer to [14] for additional rules.

**Notations.** Let  $U$  be a set of attributes  $A_1, \dots, A_n$  with domains  $dom(A_i)$ . Let  $DB$  be a database whose  $D$  is the set of all attribute domains. Let  $ID$  be the set of id-attributes of  $DB$ . The database schema is defined as a set of relation schemas  $R_1, \dots, R_m$  with  $R_i \subseteq U$ . We denote by  $PKEYS(U)$ , the set of primary Keys and by  $FKEYS(R_i, R_j)$  the set of foreign keys in  $R_i$  to  $R_j$ . Furthermore, we choose the Domain relational calculus (DRC) to represent user queries [21]. Let  $\delta_1$  be the mapping that represents matchings between relation names of and ontology concepts called *relation-concepts*;  $\delta_2$  be the mapping that represents matchings between attribute names and ontology concepts called *attribute-concepts*, and  $\delta_3$  be the mapping that represents matchings between database values and ontology concepts called *value-concepts*. Finally, let  $\delta_4$  be the mapping that represents matchings between a pair of attribute names and ontology relationship-types.

#### 4.1 Augmentation Rules

The goal of these rules is to extend the query answer with results that meet user's expectations. To this end, we have developed four rules: a Vocabulary-, a Support-, a Feature, and a Part-Whole rule [13, 14]. The first rule addresses semantic ambiguities discussed in section 2. The second rule is based on semantics of the relationships from which the ontology is constituted. The third rule is based on the domain-specific relationships that are mapped to the database model. In the following, we describe the fourth rule in details.

The basic idea of the Part-Whole rule is the use of the "part-whole" properties to discover new database objects which are closely related to those the given query returns. Based on the semantic relationship "PartOf" the rule rewrites a user query by substituting the query terms by other semantically equivalent ones. For this rule, the concepts corresponding to the substituted terms together with the "PartOf"-relationships spec-

ify the same concepts corresponding to the original query terms. Thus, the same type of the object specified in the query can be defined in another way by using an alternative set of terms. A formal description of the rule is given in Appendix B.

For example, if a user wants to retrieve data about the Item "pc" from the database  $DB_1$ , the query submitted may look like

$$Q_1 = \{(x_1, x_2, x_3, x_4) \mid (x_1, x_2, x_3, x_4) \in ITEM \wedge x_2 = "pc"\}.$$

This query asks for objects of type "pc". According to the ontology  $O_1$  we deduce that a "pc" is composed out of three parts: a "desktop", a "monitor" and a "keyboard". Assuming that all PC-objects in the database are composed exactly out of these parts, which do not participate in the composition of any other object, enables the identification of PCs by means of their components. Thus, the set of terms {"desktop", "monitor", "keyboard"} and the term "pc" are semantically equivalent.

By applying the Part-Whole rule to the query  $Q_1$  we obtain a reformulated query  $Q_1$  that retrieve also objects whose parts are the previous components. A formal description of  $Q_1$  is given in Appendix B. Therefore, it is not surprising that the tuples 123 and 128 with attribute values "computer" and "product" meet fully the intention of the user. When a user poses the query  $Q_1$  to the  $DB_1$  database, these tuples will certainly be missed. As a result, the number of tuples will increase.

## 4.2 Reduction Rules

The main feature of these rules is that after reformulating a user query the number of tuples in the answer might decrease compared to that number of tuples before any reformulation.

In the following, we describe one of such rules. We call this rule "*the sensitivity rule*" because its goal is to increase the *sensitivity* of a user query. A query is called *sensitive* if its answer contains as few as possible *false positives*. We define a tuple as *false positive* if it is semantically not correct w.r.t. the user's expectations.

For example, a problem might occur when querying a database containing homonymous terms. If a user queries a database using terms in his query expression that might be homonymous with some other terms in that database, the answer to his query might contain tuples that are irrelevant to him. For instance, the term "bank" has different meanings. It means either a container for keeping coins or a piece of furniture for sitting on or a financial institution for saving money [1]. Therefore, if a user queries a given database for information concerning an object "bank", the database might return tuples containing data about furniture, containers and institutions of type bank. This might not meet the user's intention if the user expects data only on furniture.

To solve this problem, we propose a reformulation rule based on the use of an ontology associated with the given database. By applying this rule a context could be specified for a user query. That is, the context defined by the semantic description of the data, which uses vocabularies from the ontology to express the user's intention. The intuition is to specify user queries sufficiently to derive the relevant meaning based on the ontology concepts. Thus, in the example above, the user's intention to find information about "bank" as furniture can be specified by domain specific ontologies which can

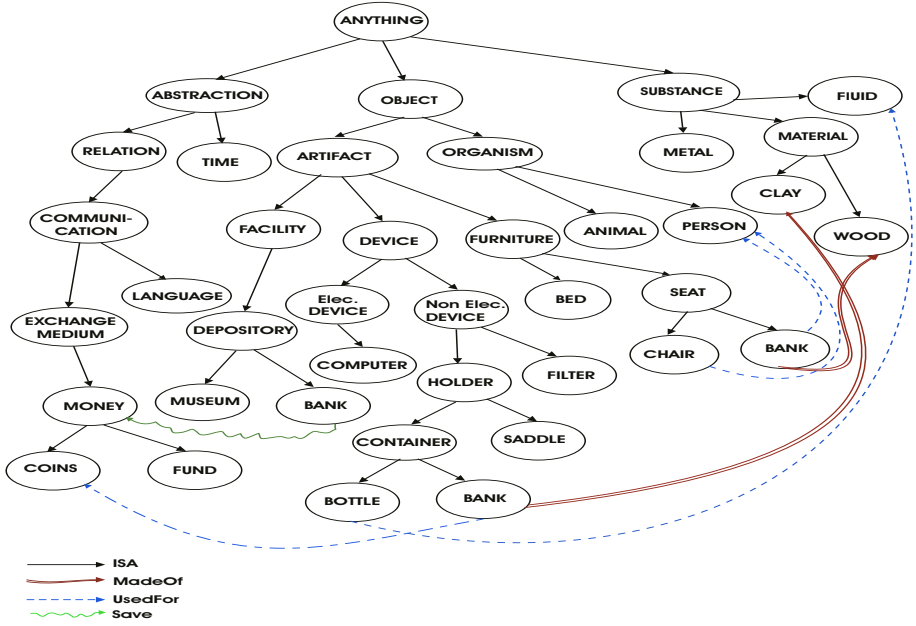


Fig. 3. Entity Ontology  $O_2$

describe different aspects of furniture. Thus, the context of user queries is restricted to furniture. However, if the ontology is more general i.e. specifies more than one context (see Figure 3). In this case it would be difficult to determine the user’s intention immediately. For example, the concept BANK might label two different nodes in two different subgraphs of the ontology. Each subgraph represents the related context of "bank". We suggest that the system asks the user to specify a unique "context". This could be done by providing him with the possibility to choose one of the ontology contexts in terms of the *immediate uncommon concepts* (imuc) of the BANK nodes. The immediate uncommon concepts of two given concepts are defined in terms of the *least common concept* (lcc) as follows:

**Definition 1.** Let  $a, b, l$  be concepts of  $\zeta$ .  $l$  is a *least common concept* (lcc) of  $a$  and  $b$  iff

- $a \in DESC("ISA", l)$  and  $b \in DESC("ISA", l)$ ,
- $\forall k, k' \in \zeta$ , if  $a, b \in DESC("ISA", k) \cap DESC("ISA", k')$  then  $k = k'$
- if  $\exists c' \in \zeta \mid a \in DESC("ISA", c')$  and  $b \in DESC("ISA", c')$  then  $l \in DESC("ISA", c')$

□

**Definition 2.** Let  $a, b, m, m', g$ , and  $g'$  be concepts of  $\zeta$ .  $m$  and  $m'$  ( $m \neq m'$ ) are *immediate uncommon concepts* (imuc) of  $a$  and  $b$  resp. iff

- $\exists l \in \zeta \mid l = lcc(a, b)$  AND
- $m = RChild("ISA", l) \wedge m' = RChild("ISA", l)$

For example, the immediate uncommon concepts of the concepts BOTTLE and CHAIR are the concepts DEVICE and FURNITURE, respectively, since their least common concept is the concept ARTIFACT.  $\square$

Next, we illustrate the sensitivity rule and its effectiveness by means of an example. A formal description of the rule is given in Appendix B.

**Example 2.** We assume a database  $DB_2$  containing information about store items. The  $DB_2$  schema might have a relation, called 'Store', whose schema defines the name of each object, the material it is made of, its use and its price. An instance of  $DB_2$  and a description of the relation 'Store' are given as follows:

**STORE(A-ID, Name, Made, Use, Price)**

A-ID: Store identifier

Name: Store name

Made: Material type

Use: Purpose of use

Price: Item price

Primary-Key(A-ID)

In addition, we assume an ontology, denoted by  $O_2$ , which describes concepts of things. A portion of  $O_2$  is adopted from [17, 6]. This ontology contains additional domain relationships: "MadeOf", "UseFor" and "Save". The meaning of "UseFor"-relationship, for example, is that if A (a concept) relates to B (a concept) by this relationship, the objects referred to A are used for purposes given by the objects referred to B. Figure 3 shows a graph representation of a portion of  $O_2$ . For the sake of clarity we omit some nodes and the other kinds of relationships.

Now, suppose that the user wants to retrieve all tuples from  $DB_2$  concerning the container 'bank'. His query can be represented as following:

$$Q_2 = \{(x_1, x_2, x_3, x_4, x_5) \mid (x_1, x_2, x_3, x_4, x_5) \in STORE \wedge x_2 = "bank"\}.$$

Obviously, the answer from the current  $DB_2$  database to the query  $Q_2$  contains the tuples 42 and 47. However, the tuple 42 does not meet the intention of the user since it relates to furniture. By using the ontology  $O_2$  the system could deduce that "bank" is related to three different contexts: Furniture, device and facility. This is done by retrieving the *imuc* of BANK concepts. Therefore, it has to ask again the user for specifying his query providing him the three relevant variants. If the user means a device "bank", the system will be able to specify the concept BANK from  $O_2$  that the related objects are used for keeping coins. Thus, the user query should include terms represented by the concept COINS to assert the intended context of the answer. The application of the rule 2 to the query  $Q_2$  leads to the following query:

$$Q_2 = \{(x_1, x_2, x_3, x_4, x_5) \mid (x_1, x_2, x_3, x_4) \in STORE \wedge (x_2 = "bank" \wedge x_3 = "coins")\}.$$

The answer to this reformulated query will contain then only the tuple 47 as expected by the user.  $\square$

## 5 Semantic Model and Criteria

In this section, we propose a semantic model and two basic criteria which allow us to validate the reformulation rules and to ensure the consistency of the ontology with its underlying database. Due to the lack of space we will not describe the validation of the proposed rules, please refer to [14] for this issue.

### 5.1 Semantic Model

The semantic model is stated as an extension of the given ontology, denoted by  $O^*$ , which includes new concepts and additional relationship types. The new concepts represent relation names, attribute names and attribute values of the database unless they already exists. We denote these concepts by  $NC_{RN}$ ,  $NC_{AN}$  and  $NC_V$ , respectively. We call *id-concepts* the concepts that represent id-values of the database and denote its set by  $\Omega$ .

The additional relationships have to relate the new concepts to the existing ones or to each other. Their types are defined as follows:

- **"ValueOf"** is the type of relationship that relates each value-concept to its associated attribute-concept.
- **"HasA"** is the type of relationship between relation-concepts and attribute-concepts.
- **"InstanceOf"** is the type of relationship that relates an Id-concept to its associated relation-concept.
- **"TupleVal"** is the type of relationship that relates value-concepts to each other, which are associated with a particular tuple.

Figure 4 shows a portion of the semantic model  $O_1^*$  related to the ontology  $O_1$  and the database  $DB_1$ .

In summary, the extended ontology is defined by  $O^* = \{G^*, \zeta^*, \mathfrak{R}^*, \mathfrak{S}^*\}$  where  $\zeta^* = \zeta \cup NC_V \cup NC_{AN} \cup NC_{RN}$ ,  $\mathfrak{R}^* = \mathfrak{R} \cup \{"ValueOf", "InstanceOf", "HasA", "TupleVal"\}$ , and  $\mathfrak{S}^*$  consists of all logical axioms related to  $\mathfrak{R}^*$ .

An extended ontology could also be expressed in a logical language. For instance, using the First Order Language (FOL)  $O^*$  can be defined as a theory  $\Gamma$  which consists of an Interpretation  $I$  and a set of well formed formulas [18].  $I$  is specified by the set of individuals  $\zeta^*$  and an interpretation function  $\cdot^I$ . Appendix C shows a logical interpretation of  $O_1^*$ .

### 5.2 Consistency Criteria

The basic consistency criteria are *correctness* and *completeness*, which aim at asserting the soundness of our framework. Hence, a set of constraints must be checked for applying correctly the transformation rules. These constraints affect the design of the ontology and the implementation of database instances. Note that the ontology must not be created from scratch but a preexisting one could be reused and adapted to the underlying database by respecting these constraints [20]. Similarly, database instances must satisfy the constraints specified by the ontology.

In order to formally define the consistency criteria we need the graph operator *SelectRel* (see Appendix A). From a semantic point of view, if two id-concept nodes

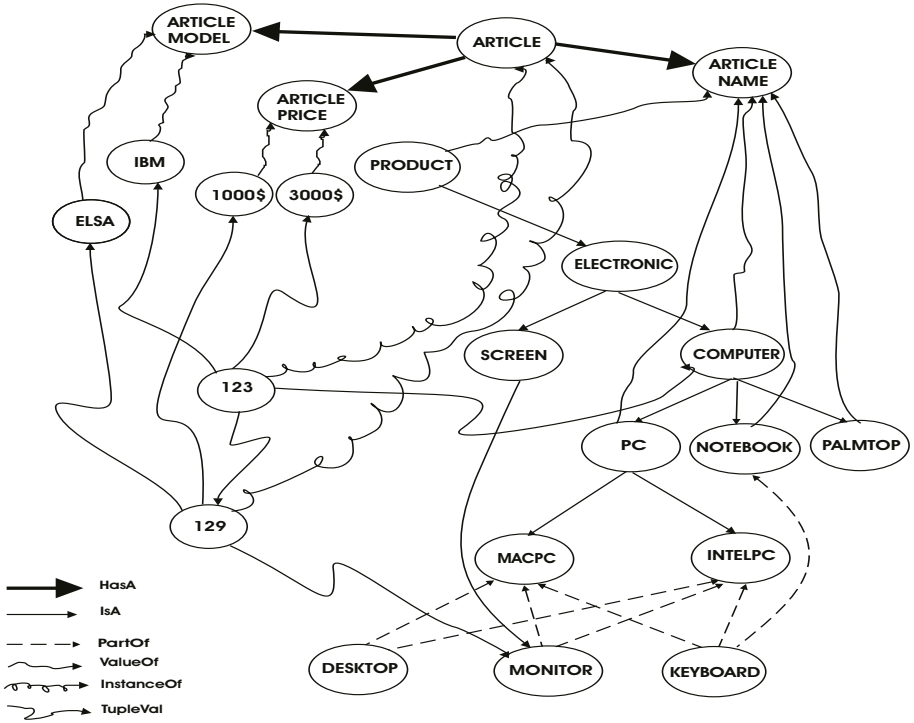


Fig. 4. A portion of the Semantic Model  $O_1^*$  in Figure 1

are adjacent (common edges are of type "TupleVal") then the semantic relationship between the represented concepts can be deduced from the result of the *SelectRel* operation on these nodes. For example, if we apply *SelectRel*-operator on the concept nodes corresponding to the identifiers 123 and 129, we can deduce that the object identified by 129 is part of the object identified by 123. We denote by  $|SelectRel_{PartOf}()|$  the number of "PartOf"-labels returned by the *SelectRel*-operator.

**Definition 3.** Let be  $ic_1, ic_2 \in \Omega$ .  $ic_1$  and  $ic_2$  are said to be *semantically dependent* if and only if  $SelectRel(G^*, ic_1, ic_2) \neq \emptyset$ .

**Correctness Criterion.** Intuitively, *correctness* means that any results of the reformulated query, say  $Q'$ , can be "derived" in the extended ontology  $O^*$  i.e. the concepts and relationships corresponding to database objects in the results of  $Q'$  must be correctly represented in the model  $O^*$ .

Formally, an extended ontology  $O^*$  is a correct model if and only if:

$\forall id_1, id_2 \in dom(ID), R \in DB, ID \in PKEYS(R)$  such that  $(id_1, c_1) \in \delta_3$  and  $(id_2, c_2) \in \delta_3$ :

1. IF  $G^*(c_1, "TupleVal", c_2)$  THEN  $c_1$  and  $c_2$  are semantically dependent
2. IF  $|SelectRel_{PartOf}(G^*, c_1, c_2)| \neq \emptyset$  THEN  $|SelectRel_{PartOf}(G^*, c_1, c_2)| = 1$

3. IF  $\exists A_i, A_j \in R \mid (\{(A_i, A_j)\}, \beta_0) \in \delta_4$  THEN  $\forall \mu \in R, G^*(c_i, \beta_0, c_j)$   
 where  $\beta_0 \in \mathfrak{R}, i \neq j, (\mu[A_i], c_i) \in \delta_3$  and  $(\mu[A_j], c_j) \in \delta_3$   $\square$

The intuition behind the first constraint is that if two database tuples are semantically related, then there exist in  $O^*$  at least one semantic relationship between the two value-concepts associated with two attribute values of the tuples. The intuition behind the second constraint is that only a `PartOf`-relation level is allowed for all the database instances i.e. if item A is part of item B and item B is part of item C than the database does not store explicitly the relation: Item A is part of item C. The third constraint asserts that if a semantic relationship between two concepts representing two attribute names exists then the concepts representing the attribute values should be related to each other through the same relationship.

**Completeness Criterion.** Intuitively, *completeness* ensures that any tuple that is "derived" in  $O^*$  for a given query  $Q'$  should also be in the answer of  $Q'$  i.e. the value-concepts together with their relationships corresponding to the results of  $Q'$  at the semantic level must be reflected in the database instance. Completeness constraints are formally described as follows:

1.  $\forall id_1 a_v p \exists id_2 Key(id_1) \wedge TUPVAL(id_1, a_v) \wedge PARTOF(a_v, p) \rightarrow$   
 $TUPVAL(id_1, id_2) \wedge TUPVAL(id_2, p) \wedge Key(id_2)$
2.  $\forall id_1 a_v p \exists id_2 Key(id_1) \wedge TUPVAL(id_1, a_v) \wedge COMMONPART(a_v, p) \rightarrow$   
 $Key(id_2) \wedge TUPVAL(id_1, id_2) \wedge TUPVAL(id_2, p)$   $\square$

Due to limited space we describe the predicates of the above formulas in Appendix C. The first axiom asserts that each decomposition of a concept in the ontology must reflect the same decomposition for its associated value in the database instance. For example, each instance of the  $DB_1$ -database where the Item name is "pc" should have "desktop", "monitor" and "keyboard" instances. In addition, this condition asserts when the `PartOf`-relationship is transitive with respect to the `ISA`-relationship. A concept, say B, is a part of a concept, say A, if B is a part of all the sub-concepts of A. For example, the concept `MONITOR` is a part of the concept `PC` because it is a part of both concepts `MACPC` and `INTELPC`, which are sub-concepts of `PC`. On the other hand, the second axiom asserts that if all the sub-concepts of A (a concept) have a common part P (a concept) then each DB-instance reflecting A must be related to an instance which reflects P.

## 6 Mappings Between Ontologies and Databases

In order to accomplish the query reformulation task, mapping information between the ontology and the underlying database must exist. This information links the concepts and the relationships of the ontology with the database elements: Relations, attributes, attribute domains.

In this section, we focus on how to define these mappings rather than how to find them. Regarding the creation of mappings there is currently no automatic method for solving this issue but semi-automatic methods based on linguistic matchings might be adequate for this purpose [10]. In the following, we define the necessary properties that make such mappings adequate for applying the transformation rules. Moreover, we specify the necessary conditions for each kind of mappings that must be verified for maintaining the consistency between the ontology and its associated database. We address each aspect of the mappings separately.

### 6.1 Mapping Between Attribute Values and Concepts

We define a simple one-to-many mapping  $\delta_3$  for each value from the set of attribute domains  $D$ . The semantic of this mapping is that each value might be represented by a single or multiple concepts, but a given concept might represent at most one value. For example, the concept COMPUTER, in the  $O_2$ -ontology, is mapped to the value "computer" of the attribute Name in the relation Item. However, if a value has multiple homonyms, it might be represented by multiple concepts.

Formally, let  $A$  be an attribute name. We define  $\delta_3$  as a relation between  $\zeta$  and  $D$  :

$$\delta_3 \subseteq D \times \zeta. \text{ Then, } \forall v_0 \in \text{dom}(A) \exists c_0 \mid (v_0, c_0) \in \delta_3 \text{ and } \delta_3 \text{ is injective.} \quad \square$$

In this context, each tuple in a given relation may be mapped to more than one concept. For example, tuple 43 in Table 3 can be mapped to two concepts related to the attribute values "chair" and "wood".

### 6.2 Mapping Between Attribute Names and the Ontology

Now, we define the mapping of attribute names to concepts and relationships of the ontology. Like the previous definitions, each attribute name might be mapped to one or more concepts in the ontology and each concept covers at most one attribute name. This mapping is also *injective*. In addition, if such mapping exists then the following constraints must be satisfied: Each value of the domain of that attribute must be mapped to a concept in the ontology. This concept must be related to the concept representing the attribute through the "ISA"-relationship.

Formally, let  $U$  be a set of attribute names. We define  $\delta_2$  as a relation between  $U$  and  $\zeta$ :  $\delta_2 \subseteq U \times \zeta$ . Then, The following conditions must be satisfied:

**Table 3.** Store relation

A-ID	Name	Made	Use	Price
41	bed	wood	kid	120 \$
42	bank	wood	person	300 \$
43	chair	wood	person	150 \$
44	flat iron	substance	clothes	60 \$
45	chain	gold	women	850 \$
46	perfume	roses	women	85 \$
47	bank	clay	coins	50 \$
48	cage	metal	birds	300 \$



- (i) IF  $\delta_2(A) = c_0 \in \zeta$  THEN  $\forall x \in \text{dom}(A), \exists c \in \zeta \mid (x, c) \in \delta_3$   
and  $c \in \text{DESC}(\text{"ISA"}, c_0)$  (1)  $\square$

Furthermore, two attribute names, say  $A_1$  and  $A_2$ , could be mapped to a single relationship-type in the ontology. The semantic of this mapping is that each concept corresponding to a value of  $A_1$  must be related to a concept corresponding to a value of  $A_2$  through this relationship.

Formally, we define  $\delta_4$  as a relation:  $\delta_4 \subseteq (U \times U) \times \mathfrak{R}$ . Then,

IF  $(\{(A_1, A_2)\}, \beta_0) \in \delta_4$  THEN

- (i) condition (1) holds for  $A_1$  and  $A_2$  and  
(ii)  $\forall x \in \text{dom}(A_1), \exists y \in \text{dom}(A_2) \mid \exists (c_x, \beta_0, c_y) \in G$   
where  $(x, c_x) \in \delta_3, (y, c_y) \in \delta_3$ , and  $\beta_0 \in \mathfrak{R}$ .  $\square$

### 6.3 Mapping Between Relations and the Ontology

So far, we presented the mappings for attributes and attribute values. Now, we address the mapping from a given relation in the database to concepts and relationships in the ontology. Like previous mapping types, a relation name might be mapped to several concepts. This mapping is also *injective*. We define two kinds of mappings: *Complete* and *partial* mappings.

The mapping is called *partial* if there exists a single concept representing the relation name and at least one concept representing an attribute name of this relation. The latter concept must be related to the concept corresponding to the relation name through the "ISA"-relationship. On the other hand, the mapping is called *complete* if all attribute names of the relation (except the ID-attribute if it is generic) are represented in the ontology and satisfy the constraint above.

Formally, let  $R$  be a relation,  $U(R)$  be a set of its attributes. We define the mapping  $\delta_3$  as a relation between  $\{R_1, \dots, R_n\}$  and  $\zeta$ :  $\delta_3 \subseteq \{R_1, \dots, R_n\} \times \zeta$ .

Let  $c_0 \in \zeta \mid (R, c_0) \in \delta_3$ . Then,  $\delta_3$  is *complete* iff:

- (i)  $\forall A \in U(R) \mid \exists c \in \zeta, (A, c) \in \delta_2$  and  $c \in \text{DESC}(\text{"ISA"}, c_0)$ .  $\square$

### 6.4 Additional Constraints for Mapping Attribute Values

In this section, we formulate a set of constraints to ensure that a semantic model  $O^*$  remains correct when introducing new concepts and relationships in the ontology  $O$  to represent database values which are not already represented in  $O$ .

So far, if  $O$  does not cover an attribute, say  $A$  i.e. there exists a set of attribute values of  $A$ , say  $V_0$ , which are not represented by concepts in  $O$ , then new concepts should be created in  $O$ . To this end, we propose the following principles: for each  $v_0 \in V_0$ ,

- create a new node  $n_0$  in  $G$  with label  $l: (v_0, l) \in \delta_3$ ,
- if a node  $n$  exists that corresponds to  $A$  such that  $(A, n) \in \delta_1$  then relate  $n_0$  with that node using an edge of type "ISA". Otherwise, relate it with the universal concept node using the same edge type.  $\square$

So far new concepts are introduced in  $O$ , relationships among them and between existing concepts should be determined. These relationships are specified using the map-

ping information defined between attribute pairs and ontological relationship-types. To this end, each tuple in the database, in which a value  $v_0$  of  $V_0$  appear is examined as follows:

Let  $\mu$  be a tuple of a relation  $R$ , and  $U(R) = \{A_0, A_1, \dots, A_n\}$  so that  $v_0 = \mu[A_0]$ . If there exists  $A_k \in U(R)$  such that  $(\{(A_0, A_k)\}, \beta_0) \in \delta_4, \beta_0 \in \mathfrak{R}$ , then:

- insert edges of type  $\beta_0$  between the node corresponding to  $v_0$  and the children nodes of the node corresponding to  $\mu[A_k]$  (w.r.t. the ontology design choice).
- if the node corresponding to  $\mu[A_k]$  has no children then insert one edge of type  $\beta_0$  between the node corresponding to  $v_0$  and that node corresponding to  $\mu[A_k]$ .  $\square$

Concerning the problem of homonyms, the intervention of an ontology expert is needed for this task.

## 7 Conclusion

Recently, there is a growing interest in ontologies for managing data in database and information systems. In fact, ontologies provide good supports for understanding the meaning of data. They are broadly used in information integration systems to overcome problems caused by the heterogeneity of data and to optimize query processing among the distributed sources. In this paper, we use ontologies within a single relational database and present an approach of query processing using semantic knowledge from a given ontology to reformulate a user query in such way that the query answer is meaningful to the users. To this end, we propose a set of query reformulation rules and illustrate their effectiveness by some running examples. Furthermore, we present a semantic model and two basic criteria to prove the soundness of our approach. We also illustrate the semantic of mappings between the ontology and the database.

In the future work, we intend to design and develop a prototype based on this approach. To this end, we attempt to reuse an existing ontology and adopt it with an associated database with respect to our framework. In addition, we intend to extend our approach to enable the use of the semantic rules in federated database systems.

## References

- [1] Online english dictionary. [www.onelook.com](http://www.onelook.com), 2005.
- [2] R. B. A. Borgida. Conceptual modeling with description logics. *The Description Logic Handbook - Theory, Implementation and Applications*. pp: 349-372. Cambridge University Press, 2003.
- [3] S. Bergamaschi, C. Sartori, D. Beneventano, and M. Vincini. ODB-tools: A description logics based tool for schema validation and semantic query optimization in object oriented databases. *Advances in Artificial Intelligence*, 5th Congress of the Italian Association for Artificial Intelligence, Rome, Italy, 1997.
- [4] B. Chandrasekaran, J. Josephson, and V. Benjamins. What are ontologies, and why do we need them? In *IEEE Intelligent Systems*, pages 20–26, 1999.

- [5] M. Dzbor, J. Domingue, and E. Motta. Magpie- towards a semantic web browser. In *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 690–705, 2003.
- [6] C. Fellbaum. Wordnet an electronic lexical database, 1998.
- [7] T. Gruber. A translation approach to portable ontology specifications. In *Knowledge Acquisition (5) No. 2, USA*, pages 199–220, 1993.
- [8] N. Guarino and P. Giaretta. Ontologies and knowledge bases: towards a terminological clarification. In *Knowledge Building Knowledge Sharing, ION Press*, pages 25–32, 1995.
- [9] J. Heflin and J. Hendler. Dynamic ontologies on the web. In *AAAI/IAAI*, pages 443–449, 2000.
- [10] M. Hernandez, R. J. Miller, and L. M. Haas. Clio: A semi-automatic tool for schema mapping. In *ACM SIGMOD*, 2001.
- [11] A. Kaye and R. Colomb. Extracting ontological concepts for tendering conceptual structures. *Data and Knowledge Engineering*, 41(1-4), 2001.
- [12] E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Conference on Cooperative Information Systems*, 41:14–25, 1996.
- [13] C. B. Necib and J. Freytag. Ontology based Query Processing in Database Management Systems. In *Proceeding on the 6 th international conference on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems (ODBASE'2003)*, pages 37–99, 2003.
- [14] C. B. Necib and J. Freytag. Using Ontologies for Database Query Reformulation. In *Proceeding on the 18 th conference on Advances in Databases and Information Systems (ADBIS'2004)*, 2004.
- [15] N. Noy and C. D. Hafner. The state of the art in ontology design. *AI Magazine*, 3(18):53–74, 1997.
- [16] N. Paton, R. Stevens, P. Baker, C. Goble, S. Bechhofer, and A. Brass. Query processing in the TAMBIS bioinformatics source integration system. *Statistical and Scientific Database Management*, pages 138–147, 1999.
- [17] A. Pease. The sigma ontology development environment. In *IJCAI-03 Workshop on Ontologies and Distributed Systems (ODS'03), Acapulco, Mexico*, Lecture Notes in Computer Science, 2003.
- [18] M. Peim, E. Franconi, N. Paton, and C. Goble. Query processing with description logic ontologies over object-wrapped databases. technical report, University of Manchester, 2001.
- [19] A. Perez and V. Benjamins. Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*, 1999.
- [20] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data Knowledge Engineering*, 25(1-2):161–197, 1998.
- [21] J. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, 1988.
- [22] H. Wache, T. Voegelé, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Huebner. Ontology-based integration of information- a survey of existing approaches. In *Proc. of IJCAI*, 2001.

## A Graphical Operations

A set of primitive graph operations: *ISACHild*, *RChild*, *RParent*, *ANCES*, *DESC*, *SUBT*, *SYNs* and *PARTs*, are needed for formal representations of the transformation rules.

Let  $P_{ths}(c_i - c_j)$  be a set of directed paths between two concept nodes  $c_i$  and  $c_j$ . Let be  $c_1, c_2, s_k, s_h \in \zeta$  and  $R, R_i \in \mathfrak{R}$ :

- $ISACHild(c_1) = \{c_2 \mid G(c_1, "ISA", c_2)\}$
- $SYNs(c_1) = \{c_2 \mid G(c_1, "SynOf", c_2)\}$
- $Rchild(R, c_1) = \{c_2 \mid G(c_1, R, c_2)\}$
- $Rparent(R, c_1) = \{c_2 \mid G(c_2, R, c_1)\}$
- $SUBT(c_1) = \{c_2 \in \zeta \mid \exists P_{ths}(c_1 - c_2)\}$
- $DESC(R, c_1) = \{c_2 \in \zeta \mid \exists p \in P_{ths}(c_1 - c_2) : \forall e = (s_k R_i s_h) \in p, R_i = R\}$
- $ANCES(R, c_1) = \{c_2 \in \zeta \mid \exists p \in P_{ths}(c_2 - c_1) : \forall e = (s_k R_i s_h) \in p, R_i = R\}$
- $SelectRel(G^*, c_1, c_2) = \{R_i \in \mathfrak{R} \mid \exists A, B \in V, \exists P \in P_{ths}(A, B) : G^*(c_1, "TupleVal", A), G^*(c_2, "TupleVal", B) \wedge \exists s_k R_i s_h \in P\}$   $\square$

Informally,  $ISACHild(c)$  is the set of the immediate sub-concepts of  $c$  (a concept).  $Rchild(R, c)$  is the set of all descendant concepts of  $c$  following edges of type  $R$ . Similarly,  $Rparent(R, c)$  is the set of all ascendant concepts of  $c$  following edges of type  $R$ .  $DESC(R, c)$  returns the set of all descendant concepts of  $c$  following edges of type  $R$ , whereas  $ANCES(R, c)$  returns the set of all ascendant concepts of  $c$  by following edges of type  $r$ . Similarly,  $SUBT(c)$  returns all descendants of  $c$  for any edge-type and  $SYNs(c)$  returns the set of all synonyms of  $c$ .  $SelectRel$  returns all edge types of the paths between two concepts connected with other concepts via edges of type "TupleVal".

In addition, we define an  $Outgoings(c)$  as a set of edge-types going out from the node of a concept  $c$ . We also define a  $PARTs(c)$  as a set of concepts that are "parts" of the concept  $c$ . According to our ontology graph design  $PARTs(c)$  is determined by traversing the nodes related with to  $c$  following only edges of type "PartOf" and "ISA". More precisely, two cases must be distinguished:

- Case 1: If  $Outgoings(c) \ni "PartOf"$  then  $PARTs(c) = A \cup B \cup C$  where
  - $A = DESC("PartOf", c)$
  - $B = DESC("ISA", a), a \in A$
  - $C = SYNs(h) \cup SYNs(l), h \in A$  and  $l \in B$ .

Informally,  $PARTs(c)$  is the set of concepts obtained by retrieving the labels of all nodes that are PartOf-children of the node  $c$  together with their ISA-descendants and synonyms.

- Case 2: If  $Outgoings(c) \ni "ISA"$  then  $PARTs(c) = PARTs(s_i)$  where  $s_i \in A$  and  $\forall (s_1, s_2) \in A^2 PARTs(s_1) = PARTs(s_2)$ ,  $A = DESC("ISA", c)$ .

Informally,  $PARTs$  of a concept  $c$  is defined recursively in terms of its sub-concepts. It is equal to the  $PARTs$  of one of its sub-concepts (if they have the same  $PARTs$ ).  $\square$

## B Syntax of Query Reformulation Rules

Let be  $t_0 \in D$  and  $R, R_1, R_2 \in DB$ . Part-Whole and Sensitivity rules are formulated as follows.

### B.1 Part-Whole Rule

**IF**  $Q = \{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R_1 \wedge x_i \theta t_0\}$   
 and  $\exists A_1, A_2 \in FKES(R_2, R_1) \mid \delta_4(A_1, A_2) = \text{"PartOf"}$   
 and  $\exists c_0 \in \zeta \mid \delta_3(t_0) = c_0$   
 and  $\forall c_i \in \zeta, c_i \neq c_0, PARTS(c_0) \not\subseteq PARTS(c_i)$

**THEN**  $Q = \{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R_1 \wedge x_i \theta t_0\} \cup$   
 $\{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R_1 \wedge [\exists(y_{11}, \dots, y_{n1}) \mid (y_{11}, \dots, y_{n1}) \in R_2 \wedge$   
 $x_1 = y_{11} \wedge \exists(z_{11}, \dots, z_{n1}) \in R_1 \wedge (z_{11} = y_{21} \wedge z_{i1} = s_1)] \wedge \dots \wedge [\exists(y_{1m}, \dots, y_{nm}) \mid$   
 $(y_{1m}, \dots, y_{nm}) \in R_2 \wedge x_1 = y_{1m} \wedge \exists(z_{1m}, \dots, z_{nm}) \in R_1 \wedge (z_{1m} = y_{2m} \wedge z_{im} =$   
 $s_m)]\}$

where  $s_j \in I_0 = \{t \in D \mid \delta_3(t) \in PARTS(c_0)\}, 1 = < j = < m = |I_0|$ .

By applying this rule on the query  $Q_1$  (section refaugmentation rules), the reformulated query is given as follows:

$Q_1 = \{(a_1, a_2, a_3, a_4) \mid (a_1, a_2, a_3, a_4) \in ITEM \wedge a_2 = \text{"pc"}\} \cup$   
 $\{(a_1, a_2, a_3, a_4) \mid (a_1, a_2, a_3, a_4) \in ITEM \wedge [\exists y_1, y_2 \mid (y_1, y_2) \in COMPONENT$   
 $\wedge a_1 = y_1 \wedge \exists(b_1, b_2, b_3, b_4) \in ITEM \wedge (y_2 = b_1 \wedge b_2 = \text{"monitor"})] \wedge$   
 $[\exists z_1, z_2 \mid (z_1, z_2) \in COMPONENT \wedge a_1 = z_1 \wedge \exists(c_1, c_2, c_3, c_4) \in ITEM \wedge (z_2 =$   
 $c_1 \wedge c_2 = \text{"keyboard"})] \wedge [\exists u_1, u_2 \mid (u_1, u_2) \in COMPONENT \wedge$   
 $a_1 = u_1 \wedge [\exists d_1, d_2, d_3, d_4 \mid (d_1, d_2, d_3, d_4) \in ITEM \wedge u_2 = d_1 \wedge d_2 = \text{"desktop"}]]\}$

### B.2 Sensitivity Rule

**IF**  $Q = \{x_i \mid (x_1, \dots, x_n) \in R \wedge x_p \theta t_0\}$   
 and  $\exists c_0, c_p \in \zeta \mid \delta_3(t_0) = c_0$  and  $c_p = \delta_2(A_p)$   
 and  $\exists A_i, \dots, A_j \in U(R) \mid \delta_4(A_p, A_k) = r_k \in \mathfrak{R} \setminus \{\text{"PartOf"}\}$   
 and  $c_0 \in SUBT(c_k) \cap SUBT(c_p), c_k = \delta_2(A_k)$

**THEN**  $Q = \{(x_1, \dots, x_n) \mid (x_1, \dots, x_n) \in R \wedge (x_p \theta t_0) \bigwedge_{k=i}^j (\bigvee_{h=1}^m x_k \theta t_{kh})\}$

where  $t_{kh} \in I_0 \cup I_1 \cup I_2, m = |I_0 \cup I_1 \cup I_2|$

$I_0 = \{t \in D \mid \delta_3(t) = c_{kh}\}$

$I_1 = \{t \in D \mid \delta_3(t) \in DESC(\text{"ISA"}, c_{kh})\}$

$I_2 = \{t \in D \mid \delta_3(t) \in SYNS(c_{kh}) \vee \in SYNS(a), a \in DESC(\text{"ISA"}, c_{kh})\}$

$c_{kh} \in DESC(\text{"ISA"}, c_k) \cap RParent(r_k, c_0)$ , and

$i = < k = < j = < n, k \neq p, 1 = < h = < m$ .

□

## C Logical Interpretation

$$\Gamma: I = (\zeta^*, \cdot^I)$$

$$ISA^I = \{(a, b) \in \zeta^{*2} \mid G^*(a, \text{"ISA"}, b)\}$$

$$SYN^I = \{(a, b) \in \zeta^{*2} \mid G^*(a, \text{"SynOf"}, b)\}$$

$$PARTOF^I = \{(a, b) \in \zeta^{*2} \mid G^*(a, \text{"PartOf"}, b)\}$$

$$HASA^I = \{(a, b) \in \zeta^{*2} \mid G^*(a, \text{"HasA"}, b)\}$$

$$VALUEOF^I = \{(a, b) \in \zeta^{*2} \mid G^*(a, \text{"ValueOf"}, b)\}$$

$$INSOF^I = \{(a, b) \in \zeta^{*2} \mid G^*(a, \text{"InstanceOf"}, b)\}$$

$$Key^I = \{a \in \zeta^* \mid G^*(a, \text{"InstanceOf"}, b)\}$$

$$TUPVAL^I = \{(a, b) \in \zeta^{*2} \mid G^*(a, \text{"TupleVal"}, b)\}$$

$$WHOLE^I = \{a \in \zeta^* \mid \forall b_1 b_2 c \ ISA(a, b_1) \wedge ISA(a, b_2) \wedge PARTOF(b_1, c) \rightarrow PARTOF(b_2, c)\}$$

$$\forall x. ISA(x, x)$$

$$\forall x. SYN(x, x)$$

$$\forall x. PARTOF(x, x)$$

$$\forall xyz. ISA(x, y) \wedge ISA(y, z) \rightarrow ISA(x, z)$$

$$\forall x.y SYN(x, y) \leftrightarrow SYN(y, x)$$

$$\forall xyz. SYN(x, y) \wedge SYN(y, z) \rightarrow SYN(x, z)$$

$$\forall xyz. ISA(x, y) \wedge SYN(y, z) \leftrightarrow ISA(x, z)$$

$$\forall xyz. ISA(x, z) \wedge SYN(x, y) \leftrightarrow ISA(y, z)$$

$$\forall xy \exists z. VALUEOF(x, y) \rightarrow HASA(z, y)$$

$$\forall xy \exists z. TUPVAL(x, y) \rightarrow INSOF(x, z)$$

$$\forall xyz. PARTOF(x, y) \wedge SYN(y, z) \leftrightarrow PARTOF(x, z)$$

$$\forall xyz. PARTOF(x, y) \wedge SYN(x, z) \leftrightarrow PARTOF(z, y)$$

$$\forall xyz. PARTOF(x, y) \wedge PARTOF(y, z) \rightarrow PARTOF(x, z)$$

$$\forall xyz. VALUEOF(y, z) \wedge ISA(x, y) \rightarrow VALUEOF(x, z)$$

$$\forall xyz. VALUEOF(y, z) \wedge SYN(x, y) \rightarrow VALUEOF(x, z)$$

$$\forall xyz. \exists w. INSOF(x, y) \wedge HASA(y, z) \rightarrow TUPVAL(x, w) \wedge VALUEOF(w, z)$$

$$\forall xyz. WHOLE(x) \wedge ISA(x, y) \wedge PARTOF(y, z) \leftrightarrow PARTOF(x, z)$$

$$\forall xyz_1 z_2. COMMONPART(x, y) \leftrightarrow ISA(x, z_1) \wedge ISA(x, z_2) \wedge PARTOF(z_1, y) \wedge PARTOF(z_2, y)$$

$x, y, w, z, z_1, z_2$  are variables. □

# Estimating Recall and Precision for Vague Queries in Databases

Raquel Kolitski Stasiu\*, Carlos A. Heuser, and Roberto da Silva

Instituto de Informática, Universidade Federal do Rio Grande do Sul,  
Av. Bento Gonçalves, 9500, CEP 91501-970 - Porto Alegre, RS, Brazil  
{rkstasiu, heuser, rdasilva}@inf.ufrgs.br

**Abstract.** In *vague* queries, a user enters a value that represents some real world object and expects as the result the set of database values that represent this real world object even with not exact matching. The problem appears in databases that collect data from different sources or databases were different users enter data directly. Query engines usually rely on the use of some type of similarity metric to support data with inexact matching. The problem of building query engines to execute vague queries has been already studied, but an important problem still remains open, namely that of defining the threshold to be used when a similarity scan is performed over a database column. From the bibliography it is known that the threshold depends on the similarity metrics and also on the set of values being queried. Thus, it is unrealistic to expect that the user supplies a threshold at query time. In this paper we propose a process for estimation of recall/precision values for several thresholds for a database column. The idea is that this process is started by a database administrator in a pre-processing phase using samples extracted from database. The meta-data collected by this process may be used in query processing in the optimization phase. The paper describes this process as well as experiments that were performed in order to evaluate it.

## 1 Introduction

In *vague* queries, the problem is to find all database values that represent the same real world as the one represented by the value entered by the user in the query. A vague query accept variation in spelling to consider not exact match in query argument compared to the values in the database. This type of query is usual in databases that collect data from different sources or are generated by different users. As an example consider a query like “Authors that have published at the ‘Intl. Conf. on Very Large Databases’ in 2002”. In the database the name of this conference may be spelled in different ways, like ‘International Conference on Very Large Databases’ or simply ‘VLDB’.

For this kind of problem a typical Information Retrieval (IR) solution is to *rank* the values in the database using some type of similarity metric (e.g. an edit distance [1, 2])

---

\* On leave from Pontifícia Universidade Católica do Paraná ([www.pucpr.br](http://www.pucpr.br)) and Centro Federal Tecnológico do Paraná ([www.cefetpr.br](http://www.cefetpr.br)).

and display them to the user ordered according to this ranking. The values that are more similar to the query value should be shown first. The user pages through the output identifying the database values that represent the real object that is being searched.

However, not all queries can be solved by applying a single similarity search over the database. In more complex queries the values resulting from one similarity search are used as query values for another similarity search. For example, consider a database that contains two tables with data collected from the Web: `ConfPaper` (`ConfName`, `PaperTitle`) and `PaperAuthor` (`PaperTitle`, `AuthorName`). If both tables are fed with data from different sources it may happen that the title of one specific paper is spelled differently in both tables. In order to process the example query above, two similarity searches must be executed: (1) over table `ConfPaper` retrieving the paper titles for the conference name given in the query and (2) over table `PaperAuthor` retrieving the author names for the paper titles found in step (1) (actually this last step is a similarity join operation [3, 4, 5, 6]).

The classical database solution for processing such types of queries is to build a query *execution plan*. An execution plan defines the query operators that are used in each step as well as the order of execution of those operators. In the context of similarity queries additionally to the classical database operators (table scan, index scan, join, ...), similarity operators like a similarity table scan [3], similarity index scan [7] and a similarity join [4, 5] are involved. The problem of building query execution plans for this type of queries has already been addressed in query systems that handle vague queries, like Vague and Query Refinement System Architecture [8, 9].

An open problem in these systems is to automatically determine the results that are relevant for a query. The IR approach described above presents all results to a user and leaves to him the task of picking up relevant results. This approach is not feasible if we are handling large data sets. The entire set of database values would appear as result of each step of the execution plan, leading to unacceptable performance. Thus, *thresholds* must be established for each similarity operator involved in the query execution plan.

The threshold to be applied depends on factors like the specific similarity metric that is being applied and the set of values to be queried. The threshold will depend also on the quality of the result that the user expects stated, for example, in classical IR measures like recall or precision [10].

In this paper we focus on the problem of semi-automatically estimating recall and precision for queries on a specific database column when a specific similarity metric is applied. The user intervention required in our approach is small. The user must just provide an approximation of the number of different real world objects that are represented in a small sample (typically 50 values) of the database column. Using this number as input and applying the process described in this paper the database system may generate meta-data that contains estimations of recall and precision for queries on the column, when different thresholds and different similarity metrics are considered. This meta-data may be subsequently used during query optimization phase. It may help the query optimizer in the choice of similarity operators, similarity metrics, as well as thresholds to be used when processing a specific query.

It should be noted that the idea of gathering information about the values in the columns in a database is central to query optimization [11, 12]. The query optimizers



of many commercial DBMS depend on meta-data gathered at specific time points determined by a database administrator. In the case of the process described in this paper the database administrator would have to additionally provide the information for the recall and precision estimation process.

This paper is organized as follows. In Section 2 related work is discussed. Section 3 describes our method to estimate recall and precision values. Section 4 presents the experiments that were performed in order to validate the proposed approach. Section 5 presents the conclusion and discusses future work.

## 2 Related Work

The idea of vague or inexact queries over databases is not new and has been studied from several points of view.

**Vague or Imprecise Queries Over Database.** The Vague System [8] and the Query Refinement System Architecture [9] are examples of vague query database processing. The former discusses a query language (an extension of QUEL) and a general model of vague queries implementation over a relational database. In this system the similarity metrics (called data metrics) are user provided. The latter discusses the problem of query refinement in the presence of similarity queries. Both cited approaches assume the availability of a user provided distance measure.

Another system that implements vague queries is the Imprecise Query Engine (IQE) [13]. Here the query engine is implemented over a classical query engine to handle vague queries. The similarity query engine converts the vague query into equivalent precise queries that appear in an existing query workload.

**Probabilistic Algebra.** Dey [14] has proposed an extended relational model and new algebraic operators supporting probabilistic aspects. Fuhr [15] presents the PRA (Probabilistic Relational Algebra) which is a generalization of the relational algebra. PRA represents a logical data model allowing close integration of IR and database to model probability values, but makes no assumptions about the underlying physical data model. The WHRIL [16] system is also based on Fuhrs work and uses text-based similarity and logic-based data access as known from Datalog to integrate data from heterogeneous sources. The work of several other authors follow the same line [17, 18, 19, 20]. These proposals include the use of redefined relational operators as select, join, etc. with a probability value associated to each attribute or tuple. This value is a measure of uncertainty obtained from a probabilistic model based on preprocessing of stored data.

These approaches use probabilistic models to compute similarity values in a preprocessing phase. This is similar to our pre-processing phase to create meta-data. However, in these approaches probability values must be stored associated with tuples or attributes whereas in our approach similarity values are dynamically computed.

**Probabilistic Model and XML.** Several proposals explore the probabilistic IR model. An example considering XML documents is the TIJAH system [21], an XML-IR system

where XML documents are treated as ‘flat-text’. This query model extends XPath with a special function called *about*. TIJAH is based on region algebra [22] and it is used to rank node-set trees. The idea behind region algebra [22] is the representation of text documents as a set of extents where each one is defined by its starting and end position. Another similar approach is XIRQL [23], which develops an algebra that implements the querying capabilities found in XPath extended with probabilistic functions. This approach is different from ours because we apply specific similarity metrics for each column.

**IR-Style Ranking.** Several IR-style systems implement *k-top* queries instead of considering a rank that is cut by a threshold value [24, 25, 26].

Additionally to the study of similarity searches over data sets, the problem of *proximity joins* or *similarity joins* has also received attention. Gravano [4, 5] describes an approach for similarity based on joins on string attributes. This work is based on the identification of all string pairs (or set of strings) similar to each other using cosine similarity metric [10] with weights derived from term frequency-inverse document frequency (tf-idf) to join similar data. Cohen [27] describes WHRIL, which is also based on cosine similarity to integrate information from structured information sources that contain textual information. Cohen describes efficient algorithms to compute the top scoring matches of a ranked result set.

Cohen [28] presents a survey comparing several similarity metrics for specific domains. This work shows that the quality of the result of a query may be improved if specific domain similarity metrics are used.

Schallehn [3, 29] presents a set of redefined relational operators to process vague queries. His work shows how the operators can be used to evaluate the query using these redefined operators.

What can be generally observed in related work is that a critical point is how to specify a threshold to meet requirements regarding efficiency and accuracy [29]. This should not be done by the user because it would lead to several trial-and-errors cycles. Our approach differs also in evaluate the quality of intermediate results. None of related work studied refers to evaluate the result set produced by similarity functions specific for domains.

### 3 The Estimation Process

In this section we describe the process by which a vague query engine can estimate recall and precision for a database column.

At specific time points defined by a database administrator (DBA), traditional DBMS gathers statistics about the database (number of different values in a column, distribution of the values in a column, . . .). These statistics are used by the query engine during so called *cost optimization* [30].

In our case, the aim of this preprocessing phase is to estimate recall/precision tables for approximate queries on specific database columns. A recall/precision table contains estimated precision and recall values for several different threshold values.

For each similarity metric that may be applied to the column, a recall/precision table will be generated. These values can be used to optimize the query in the query processing phase.

The specific metrics that can be used depend on the column domain [28]. For example, a column containing **author names** and a column containing **dates** probably would require different similarity metrics. The association of similarity metrics to database columns could be established in the database schema.

Further for the same domain several different similarity metrics may be applied. For example, in a column with person names if person names are always written in the same order (e.g. name, surname) but may spelled in different ways, an edit distance metric like Levenshtein [1] is adequate. However if the words that comprise the name may appear in different orders another kind of similarity metric may be used.

Therefore, in our approach we allow several different metrics for each column and estimate recall/precision for each of them. This information may be used by the query engine to decide which of them is better suited for the specific set of values that appear in the database.

Notice that recall/precision tables need to be generated only for those database columns that may appear as arguments in vague queries.

The process of estimation is executed once for each database column and comprises the steps described below.

1. *Sampling*

A random sample of the values in the database column is generated. Our experiments have shown that a sample of 50 values is enough.

2. *DBA intervention*

The values in the sample are displayed to the database administrator (DBA). The DBA counts the number of different real world objects that are represented by the values in the sample. Remember that different values (e.g. “VLDB” and “Very Large Databases”) may represent the same real world object.

The DBA enters the number  $n_o$  of real world objects that appear in the sample in the system.

3. For each similarity metric that may be applied to the database column the following steps are performed.

- (a) *Clustering*

The values in the sample are clustered. Clustering begins with a predetermined threshold and is repeated iteratively with different thresholds until the number of clusters  $n_c$  is equal to the number of real world objects ( $n_o$ ) that were identified by the user in the sample.

The underlying idea is that, if the similarity metric behaves correctly, each cluster will contain values that represent a single real world object.

- (b) *Recall/precision computation*

The usual approach to compute precision and recall requires user intervention. In this approach queries are stated against the database and the user identifies false positives and false negatives in the result set.

In our approach we aim at minimizing user intervention. Recall/precision will be automatically computed by the procedure described below.

Each value in the sample is used as the query value. This means that with a sample of size 50, we will execute 50 queries. Each query will result in a set of ranked values.

In order to estimate recall/precision the set of objects that should result from the query must be identified. This would usually require user intervention. In our approach we will use the cluster instead in which the query value is contained as the set of values that should be returned. Therefore, we will use the clustering result instead of users intervention.

Our approach is based on the assumption that the clustering process has partitioned the sample correctly in sets such that each one contains exactly those values that represent one and only one real world object. Thus, the clusters are used as the set of values that should be returned.

This way, recall and precision are computed for several thresholds. The average value of recall/precision considering all queries is regarded as the recall/precision for the similarity metric in several thresholds.

(c) *Storage of meta-data*

Recall/precision values for each threshold and each similarity metric are stored as meta-data for usage during query optimization phase.

## 4 Experiments

In this section we describe the experiments performed in order to empirically evaluate that the estimated recall/precision values hold also for the entire database.

### 4.1 Data Sets

For the experiments two data sets were chosen. Data set *City-DS* contains city names and data set *Street-DS* contains street names. These sets were taken from a real world database that contains information about students that are candidates to enrolment in a Brazilian University. Both data sets refer to the student's address. Most of these candidates come from a single Brazilian state. In both data sets data was entered directly by the candidates themselves. Thus the names of cities and streets may appear spelled in several ways due to several factors, like misspelling, different ways of abbreviation, etc.

The main characteristics of these data sets are shown in Table 1. The number of real world objects in each data set was counted by an human expert.

**Table 1.** Main characteristics of the data sets used for the experiments

<b>Data set</b>	Number of instances in the database	Number of real world objects	Average number of instances in a cluster
<b>City</b>	10180	387	26.3049
<b>Street</b>	3500	2377	1.4724

The value distribution in both sets presents several differences:

- The *City-DS* contains relatively few (387) real world objects represented. As most of the students come from the same Brazilian state, their addresses concentrate city names of this state. Approximately 45% of the values correspond to a single real world object, the largest city in the state. In the average each city appears 26 times in the database.
- The *Street-DS* contains many different real world objects (2377) as the number of different street names is much bigger than the number of cities. In average, each street is represented 1.4 times in the database.

## 4.2 Similarity Functions and Clustering Algorithm

As similarity metrics we have applied three well known metrics for comparing strings: Levenshtein or Edit Distance (*Edit*) [1], *Guth* [31] and *N-grams* [1] with 3 characters in each gram. Additionally we have applied a similarity metric (*Acronyms*) developed in our group that is adequate for the comparison of strings that contain abbreviations and acronyms [32].

The results of all similarity metrics applied are normalized between 0 and 1.

Clustering was performed using the Hierarchical Agglomerative Clustering Method [33]. The SLINK [34] Algorithm was used to implement clustering process.

## 4.3 Sample Generation

Due to the fact that the user must count the real world objects represented by the values in a sample we need to be careful with the sample size. Very small samples are not trustworthy to represent the database content but large samples are inappropriate to user interaction.

We have experimented with two sample sizes: 50 instances for the *City-DS* and 15 instances for the *Street-DS*. As the experimental results show, a sample size around 50 values leads to correct results compared to the whole database. The values in each sample were randomly selected in the database.

## 4.4 Clustering Results

As mentioned above, our approach is based on the assumption that the clustering process has partitioned the sample correctly in sets such that each one contains exactly those values that represent one and only one real world object.

To empirically validate this assumption we extracted 40 samples (each with 50 instances) from the *City-DS*.

In each sample each cluster was compared to the set of values that a user would consider as representing a single real world object. In these 40 samples 352 cities were represented. Two types of errors of the clustering process could be identified:

- The number of clusters does not converge to the number of real world objects. Even with small variations in the threshold (0.01) either the number of clusters exceeds the number of objects in the sample or the number of clusters is lower than the

**Table 2.** Clustering **errors** found considering 40 samples clustered and validated by the user

<b>Metric</b>	Number of clusters is incorrect	Content of cluster is incorrect
<b>Edit</b>	none	none
<b>Guth</b>	1.6%	2%
<b>N-Grams</b>	0.4%	0.4%
<b>Acronyms</b>	none	none

number of objects in the sample. This probably is an indication that the similarity metric is not adequate for the set of values.

- The correct number of clusters has been identified but their content is not correct, i.e., values that should appear in one cluster appear in the other.

The number of errors found in this process was very low and is summarized in Table 2.

For *Edit* and *Acronyms* similarity metrics no errors were found. The clustering process gave exactly the same results as expected by the user.

For *Guth* and *N-gram* similarity metrics a small percentage (around 1%) of clusters were incorrectly identified.

Those results show an high percentage of correct clusters. We can conclude that it is acceptable to use the clusters for recall/precision evaluation.

#### 4.5 Recall/Precision Estimation

The other premise in which our approach is founded is that the values of recall/precision that were calculated for the sample apply also to the entire database. In this section we will show experimental results to validate this premise.

We have performed experiments with both data sets.

**Experiments with *City-DS*.** Using the *City-DS* we first executed the estimation of recall/precision by applying the aforementioned process (Section 3). More specifically the following steps were executed:

1. We took 4 samples each containing 50 values from the cities data set.
2. For each sample an human expert determined the number of real world objects (cities in this case) represented by the values in the sample. This corresponds to the user intervention that should be executed by the database administrator during the pre-processing phase.
3. The samples were clustered as described above.
4. Each value in a sample was used as a query value in that sample. For each query, recall and precision were computed by the procedure aforementioned. We took the following values for the thresholds 0.9, 0.8, 0.7, 0.5 and 0.3. This resulted in 4 tables, one for each sample, containing recall/precision values for each threshold.
5. We took the average of the samples resulting in a single table with estimated recall/precision values for each threshold.

In order to evaluate these results we compared them to the results of queries against the complete data set. Each of the values in the samples (4 samples \* 50 values per sample = 200 values) was taken as a query value against the database. Again a table plotting recall/precision for each of the thresholds above was computed. This procedure was repeated for each of the four similarity metrics.

The results are shown in Figure 1. In this figure each graph corresponds to one similarity metric. The x-axis corresponds to the thresholds and the y-axis corresponds to similarity values. The values plotted are recall and precision. Dotted lines represent actual recall/precision values obtained for the entire data set, whereas continuous lines represent the estimated recall/precision values obtained from the samples.

As can be seen in the figure, estimated recall/precision follows similar curves to actual recall/precision for all four similarity metrics.

The results show also that some similarity metrics are more adequate than others. In this case, Edit and N-Grams are better metrics, since the values of recall and precision tend to be higher and less dependent from the threshold values. Guth and Acronyms are less adequate as precision decreases faster whit smaller thresholds.

In order to evaluate how close the estimated results for the sample are to the actual results calculated over the database, we computed the Mean Square Deviation (MSD) as defined by Equation (1).

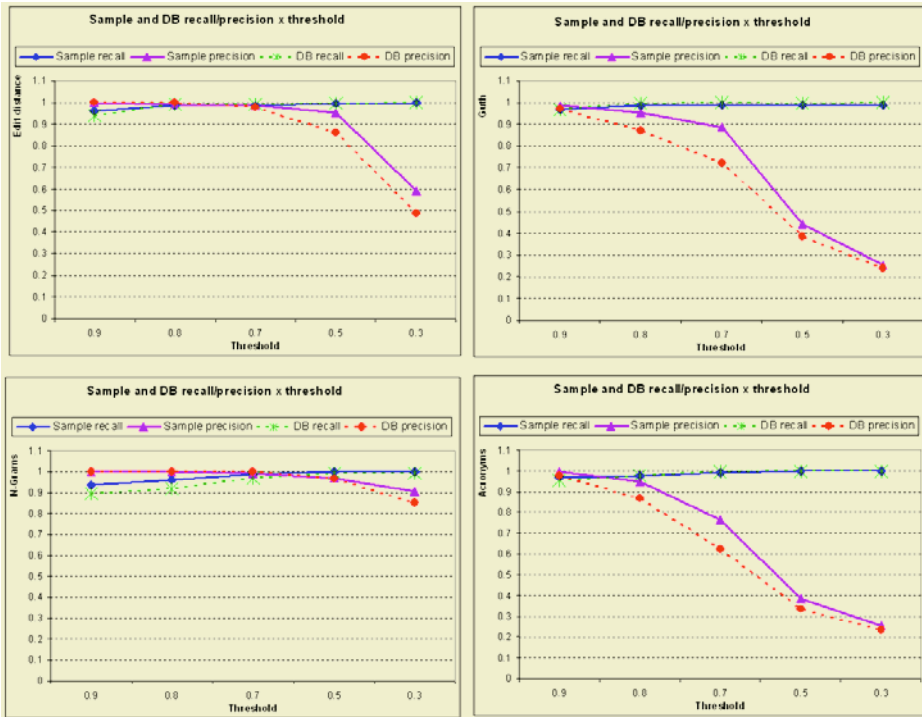


Fig. 1. City-DS – Comparing sample and database recall/precision

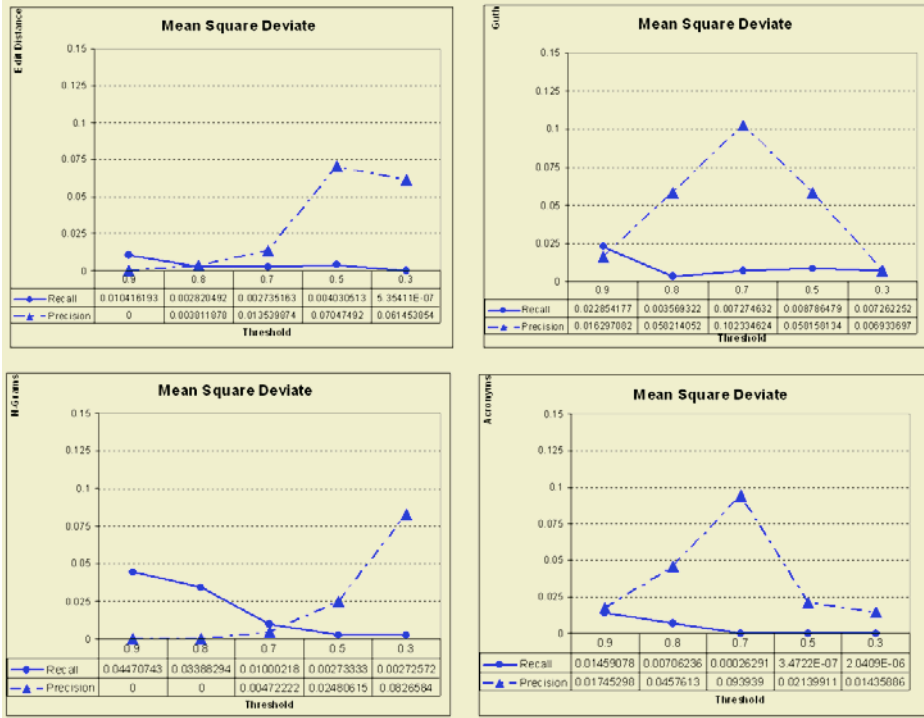


Fig. 2. City-DS: Mean Square Deviation between samples and database

$$f(x_M^b, x_M^s) = \frac{1}{n} \sum_{i=1}^n (x_{M,i}^b - x_{M,i}^s)^2, \tag{1}$$

where  $x_M^b = (x_{M,1}^b, x_{M,2}^b, \dots, x_{M,n}^b)$  is a similarity value of the database and  $x_M^s = (x_{M,1}^s, x_{M,2}^s, \dots, x_{M,n}^s)$  is a similarity value of the sample.

In Figure 2 the values of MSD obtained from Equation 1 using thresholds 0.9, 0.8, 0.7, 0.5 and 0.3 with each similarity function are shown. The values shown in that figure present the mean MSD (MSDm) for all four samples.

As can be seen the values are low showing that the estimated values are close to the actual values for the database.

We have observed that some clusters contain many duplicate values. This leads to higher similarity values. We repeated the experiment described above removing the duplicate values from the clusters. In this case similarity values are lower but still the estimated recall/precision values are similar to the actual recall/precision values for the database. Due to space restrictions the detailed results of this experiment are not shown here.

**Experiments with *Street-DS*.** To test the limits of our approach we also evaluated a data set much harder to handle, the *Street-DS*.



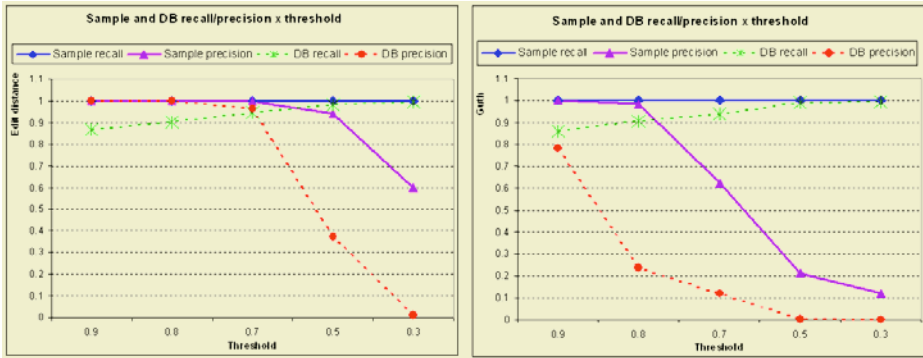


Fig. 3. Street-DS – Comparing sample and database recall/precision

The *Street-DS* was chosen because the relation between data values and real world objects is different from that in the *City-DS*. The number of real world objects is similar to that of values in the database, i.e. clusters tend to be small, many of them containing just an instance. In this case the quality of the results depends much more on the ability of the similarity metric to handle data from this domain. As few values represent each real world object every false positive or false negative changes the values of recall and precision by a considerable amount.

The results are shown in figure 3. We have used two metrics, Edit, that gave the best results in the previous experiment and Guth that gave the worst results. We have also used smaller samples (15 values) than in the previous experiment.

As can be seen in the figure, in this example the precision that was estimated is much smaller than the actual precision measured on the database. This difference is due to the inability of these similarity metrics to correctly identify which values represent the same real world object and which do not. This inability appears more clearly in the database than in the sample, because the sample contains less instances and the query value is part of those instances.

## 5 Concluding Remarks

This paper presents a contribution to the problem of query execution and optimization in a query engine that handles vague queries. Specifically we have presented an approach for estimating recall/precision values for queries with several thresholds. These values are important for query engines like that described in [8, 9].

The estimation process is to be started by a database administrator when he estimates that the distribution of values in the database has changed. We tried to minimize user intervention. The database administrator enters just a single information, namely the number of different real world objects that are represented in a small sample of the database. Based on our experiments, we can improve the sampling process through learning methods in future works.

We have described the experiments that empirically validate our approach. Specifically the experiments corroborate two premises.

- In order to estimate recall and precision we need to identify the set of values in a sample that represent a single real world object. In our approach these sets correspond to the result of the clustering process. The experiments show that the result of the clustering process may be used instead of the identification of values by a user.
- When the sample is big enough and similarity metrics are adequate for the column domain the recall/precision results are very similar to the actual recall/precision values obtained when querying the database.

However, several problems are still open.

As identified by the experiments (and also by other authors [28, 3]) some similarity metrics are more adequate than others for handling a specific column. We are working on heuristics that use the recall/precision estimations to identify what the best similarity metric for a column is.

Further, the size of the samples obviously affects the results of the estimation process. We are working on the problem of identifying what the minimum sample size for a given data set is.

## References

1. Navarro, G.: A guided tour to approximate string matching. *ACM Computing Surveys* **33** (2001) 31–88
2. Santini, S., Jain, R.: Similarity measures. *IEEE Transaction on Pattern Analysis and Machine Intelligence* **21** (1999) 871–883
3. Schallehn, E., Sattler, K.U., Saake, G.: Efficient similarity-based operations for data integration. *Data Knowl. Eng.* **48** (2004) 361–387
4. Gravano, L., Ipeirotis, P.G., Koudas, N., Srivastava, D.: Text joins in an RDBMS for web data integration. In: *Proceedings of the Twelfth International Conference on World Wide Web*, ACM Press (2003) 90–101
5. Gravano, L., Ipeirotis, P.G., Koudas, N., Srivastava, D., Muthukrishnan, S.: Approximate string joins in a database (almost) for free. In: *Proceedings of 27th International Conference on Very Large Data Bases*, September 11-14, VLDB 2001, Morgan Kaufmann (2001) 491–500
6. Schallehn, E., Geist, I., Sattler, K.U.: Supporting similarity operations based on approximate string matching on the web. In Meersman, R., Tari, Z., eds.: *CoopIS/DOA/ODBASE (1)*. Volume 3290 of *Lecture Notes in Computer Science.*, Springer (2004) 227–244
7. Navarro, G., Baeza-Yates, R.A., Sutinen, E., Tarhio, J.: Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin* **24** (2001) 19–27
8. Motro, A.: *Vague: a user interface to relational databases that permits vague queries*. *ACM Trans. Inf. Syst.* **6** (1988) 187–214
9. Ortega-Binderberger, M.: *Integrating Similarity Based Retrieval and Query Refinement in Databases*. Phd thesis, UIUC - University of Illinois at Urbana-Champaign, Urbana, Illinois (2002)
10. Baeza-Yates, R.A., Baeza-Yates, R., Ribeiro-Neto, B.: *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)

11. Ullman, J.D., Garcia-Molina, H., Widom, J.: Database Systems: The Complete Book. Prentice Hall Inc., Upper Saddle River, New Jersey, USA (2002)
12. Chaudhuri, S.: An overview of query optimization in relational systems. In: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, ACM Press (1998) 34–43
13. Nambiar, U., Kambhampati, S.: Answering imprecise database queries: a novel approach. In: Proceedings of the fifth ACM international workshop on web information and data management, ACM Press (2003) 126–133
14. Dey, D., Sarkar, S.: A probabilistic relational model and algebra. *ACM Trans. Database Syst.* **21** (1996) 339–369
15. Fuhr, N., Rolleke, T.: A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.* **15** (1997) 32–66
16. Cohen, W.W.: Integration of heterogeneous databases without common domains using queries based on textual similarity. In: Proceedings of the 1998 ACM SIGMOD international conference on Management of data, ACM Press (1998) 201–212
17. de Keijzer, A., van Keulen, M.: A possible world approach to uncertain relational data. In: 15th International Workshop on Database and Expert Systems Applications (DEXA 2004) Workshops. SIUFDB-04 1st International Workshop on Supporting Imprecision and Uncertainty in Flexible Databases, Zaragoza, Spain, September 3, 2004, IEEE Computer Society (2004)
18. Lakshmanan, L.V.S., Leone, N., Ross, R., Subrahmanian, V.S.: Proview: a flexible probabilistic database system. *ACM Trans. Database Syst.* **22** (1997) 419–469
19. Fuhr, N.: A probabilistic relational model for the integration of ir and databases. In: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval, ACM Press (1993) 309–317
20. Barbara, D., Garcia-Molina, H., Porter, D.: A probabilistic relational data model. In: Proceedings of the international conference on extending database technology on Advances in database technology, Springer-Verlag New York, Inc. (1990) 60–74
21. List, J., Mihajlovic, V., de Vries, A.P., Ramirez, G., Hiemstra, D.: The TIJAH XML-IR system at INEX 2003. Proceedings of the 2nd Initiative on the Evaluation of XML Retrieval (INEX 2003), ERCIM Workshop Proceedings (2003)
22. Consens, M.P., Milo, T.: Algebras for querying text regions: expressive power and optimization. *J. Comput. Syst. Sci.* **57** (1998) 272–288
23. Fuhr, N., Grossjohann, K.: XIRQL: a query language for information retrieval in XML documents. In: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, ACM Press (2001) 172–180
24. Fagin, R., Kumar, R., Sivakumar, D.: Efficient similarity search and classification via rank aggregation. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, ACM Press (2003) 301–312
25. Ortega, M., Chakrabarti, K., Mehrotra, S.: Efficient evaluation of relevance feedback for multidimensional all-pairs retrieval. In: Proceedings of the 2003 ACM Symposium on Applied computing, SAC 2003, ACM Press (2003) 847–852
26. Chakrabarti, K., Ortega-Binderberger, M., Mehrotra, S., Porkaew, K.: Evaluating refined queries in top-k retrieval systems. *IEEE Transactions on Knowledge and Data Engineering* **16** (2004) 256–270
27. Cohen, W.W.: Data integration using similarity joins and a word-based information representation language. *ACM Trans. Inf. Syst.* **18** (2000) 288–321
28. Cohen, W.W., Ravikumar, P., Fienberg, S.: A comparison of string distance metrics for name-matching tasks. In: Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico, Morgan Kaufmann (2003) 73–78

29. Schallehn, E., Sattler, K.U.: Using similarity-based operations for resolving data-level conflicts. In: BNCOD. (2003) 172–189
30. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In Bernstein, P.A., ed.: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, May 30 - June 1, ACM (1979) 23–34
31. Guth, G.J.: Surname spellings and computerized record linkage. *Historical Methods Newsletter* **10** (1976) 10–19
32. Dorneles, C.F., Lima, A.E.N., Heuser, C.A., da Silva, A., Moura, E.: Measuring similarity between collection of values. In: Proceedings of 6th ACM International Workshop on Web Information and Data Management (WIDM 2004), Washington DC , USA, ACM Press (2004) 56 – 63
33. Hartigan, J.A.: *Clustering Algorithms*. John Wiley and Sons, Inc., New York, NY, USA (1975)
34. Sibson, R.: SLINK: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal* **16** (1973) 30–34

# Querying Tree-Structured Data Using Dimension Graphs

Dimitri Theodoratos<sup>1</sup> and Theodore Dalamagas<sup>2</sup>

<sup>1</sup> Dept. of Computer Science,  
New Jersey Institute of Technology,  
Newark, NJ 07102  
[dth@cs.njit.edu](mailto:dth@cs.njit.edu)

<sup>2</sup> School of Electr. and Comp. Engineering,  
National Techn. University of Athens,  
Athens, GR 15773  
[dalamag@dblab.ece.ntua.gr](mailto:dalamag@dblab.ece.ntua.gr)

**Abstract.** Tree structures provide a popular means to organize the information on the Web. Taxonomies of thematic categories, concept hierarchies, e-commerce product catalogs are examples of such structures. Querying multiple data sources that use tree structures to organize their data is a challenging issue due to name mismatches, structural differences and structural inconsistencies that occur in such structures, even for a single knowledge domain. In this paper, we present a method to query tree-structured data. We introduce dimensions which are sets of semantically related nodes in tree structures. Based on dimensions, we suggest dimension graphs. Dimension graphs can be automatically extracted from trees and abstract their structural information. They are semantically rich constructs that provide query guidance to pose and evaluate queries on trees. We design a query language to query tree-structured data. A key feature of this language is that queries are not restricted by the structure of the trees. We present a technique for evaluating queries and we provide necessary and sufficient conditions for checking query unsatisfiability. We also show how dimension graphs can be used to query multiple trees in the presence of structural differences and inconsistencies.

## 1 Introduction

Tree structures provide a popular means to organize the information on the Web. Taxonomies of thematic categories, concept hierarchies, e-commerce product catalogs are examples of such structures. Since the XML language [3] has become the standard data exchange format on the Web, organizing data in tree structures has been vastly established. Even if data is not stored natively in tree structures, export mechanisms make data publicly available in tree structures to enable its automatic processing by programs, scripts, and agents on the Web [11]. Querying capabilities on these structures are provided through path

expression queries. Such queries are formed using some of the query languages proposed in the literature like XPath [4] and XQuery [5].

Querying multiple data sources that use tree structures to organize their data is a challenging issue due to name mismatches, structural differences and structural inconsistencies that occur in such structures, even for a single knowledge domain. Name mismatches appear because tree structures lack semantic information. For example, laptop computers might be referred to as **notebooks** in one product catalog but as **portables** in another catalog. In this paper, we do not focus on this issue and we assume that it is resolved using well-known schema matching techniques [20]. Structural differences and, far more important, structural inconsistencies appear because of the different possible ways of organizing the same data in tree structures. For example, a structural difference exists when a category appears in a product catalog but does not appear in another. A structural inconsistency appears when a product catalog for notebooks classifies new, SONY notebooks with 10in display in the path `/notebooks/new/SONY/10in`, while another catalog classifies the same products in the path `/SONY/notebooks/10in/new`.

A naive approach to cope with structural differences and inconsistencies when querying multiple tree structures is to generate different versions of the initial query, considering different subsets of nodes involved in its path expressions and their different orderings. Clearly this is not efficient due to the large number of queries that need to be generated. Another approach is to set up a global structure and define mapping rules between this structure and the local structures [13]. Such approaches require extensive manual effort, since the global schema is difficult to construct and the rules should be hard-coded in the application.

In this paper, we suggest a novel approach to query tree structured data. Our approach exploits semantic information for nodes of the trees which are called here value trees. We introduce the concept of a dimension that groups together semantically related values (nodes). The different dimensions of a value tree are related through precedence relationships incurred by the parent-child and ancestor-descendant relationships of their nodes. We capture these precedence relationships between dimensions of a value tree into the concept of a dimension graph for a value tree. Besides abstracting structural information of value trees, dimension graphs provide also semantic guidance in posing and evaluating queries. Query conditions involve dimensions, and thus query formulation is not dependent on the structure of value trees. The system uses the dimension graph of the value tree to identify orderings of the values that can possibly exist in the value tree. Only these value orderings will be used to compute the answer of the query on the value tree. This step of the computation of the query answer is performed before the query evaluation reaches the value tree which is, in general, much larger than its dimension graph.

**Contribution.** The main contributions of the paper are the following:

- We introduce dimensions to record semantic information for the nodes of value trees and dimension graphs to capture structural information on value trees. Dimension graphs can be automatically extracted from value trees.

- We design a query language to query value trees. Queries are not cast on the structure of a specific value tree, since they are issued on their dimensions. The user can optionally specify parent-child and/or ancestor-descendent relationships between dimensions in a query.
- We show how queries can be evaluated on value trees, making use of dimension graphs to determine orderings of dimensions that can possibly generate non-empty answers. These dimension orderings are then used for generating path expressions that are evaluated on value trees.
- We introduce the concept of query unsatisfiability which identifies a query on a dimension graph that has an empty answer on any value tree underlying this dimension graph. We provide necessary and sufficient conditions for a query to be unsatisfiable.
- Finally, we show how dimension graphs can be used to query multiple value trees in the presence of structural differences and inconsistencies.

**Outline.** The rest of the paper is organized as follows. The next section discusses related work. In Section 3, we introduce dimensions and we define dimension graphs for value trees. Section 4 presents the query language used to pose queries on dimension graphs. It also shows how queries can be checked for unsatisfiability and how they are evaluated on the underlying value trees. Finally, Section 5 concludes the paper and presents further work. Due to lack of space, proofs of propositions are omitted. They can be found in the full version of the paper.

## 2 Related Work

Many systems support query evaluation of multiple data sources, using a predefined global structure and defining mapping rules between this structure and the local structures used in the sources. The Xyleme system[13] copes with the problem of integrating XML data sources using XML views. In the Agora system[18], XQuery expressions over a given global XML schema are translated to SQL queries on local data sources. In [7], query evaluation is based on mapping rules from global to local schemas in the form of path-to-path correspondences. In [12], YAT queries are posed on a global schema and evaluated in the data sources using YAT mapping rules. In [19], Xpath queries are formed and reformulated to queries on the local catalogs, given a pre-defined DTD. Our approach differs than the aforementioned techniques in that it does not require the manual definition of hard-coded mapping rules between the virtual tree structure and the local structures.

Relevant to our work are also techniques where schema descriptions are automatically extracted from local data sources. XClust [17] generates DTDs to act as global schemas, applying clustering methods to detect similar DTDs prior to their integration. Techniques that extract DTDs from collections of XML documents are also presented in [14]. In [8], a grammar-based model is used to integrate DTDs. Contrary to our approach, these papers do not deal with query evaluation.

Schema-based descriptions for data with little or no apparent structure have also been suggested for semistructured databases [6]. Dataguides are introduced in [15]. They are structural summaries for semistructured data, useful for formulating queries, storing statistics about paths and nodes, and enabling query optimization. In [10], graph schemas are introduced to formulate, optimize and decompose queries for semistructured data. These approaches do not provide a direct solution to the problem of structural inconsistencies and differences in data sources that we address here. Further, they are purely syntactic. In contrast to our approach, they do not exploit semantic information.

Integrating tree-structured data is also a popular issue in e-commerce applications [9, 16]. Facet classification hierarchies [1, 2] also exploit sets of semantically related categories. In [21], the authors present faceted taxonomies for Web catalogs. None of these works suggest query evaluation techniques.

### 3 Data Model

In this section we present a data model for tree-structured data. We introduce a type of trees, called value trees, to represent tree-structured data. We also discuss the notion of a dimension, based on which a partitioning can be enforced on value trees.

#### 3.1 Value Trees and Dimensions

We assume a set of values  $V$  that includes a special value  $r$ . The elements of  $V$  are used to build value trees.

**Definition 1.** A value tree is a rooted node-labeled tree  $T$ , such that:

- (a) Each node label in  $T$  belongs to  $V$ .
- (b) Value  $r$  labels only the root of  $T$ .
- (c) There are no sibling nodes in  $T$  labeled by the same value. □

Figure 1 shows examples of value trees  $T_1$ ,  $T_2$  and  $T_3$  (for the moment, the dotted labeled rectangles that group the nodes should be ignored). These value trees are parts of taxonomies used to categorize products related to computer equipment. The same value may label multiple nodes in a value tree. For example, value HP labels two nodes in  $T_2$ . Notice that there are structural differences and inconsistencies between value trees  $T_1$ ,  $T_2$  and  $T_3$ , although they refer to the same knowledge domain. For example, there are nodes labeled **Multimedia** or **Servers** in  $T_2$  and  $T_3$ , even though no such nodes appear in  $T_1$ . Also, a node labeled **Used** is a child of a node labeled **Sony** in  $T_2$ , although the opposite holds in  $T_3$ . Note that we assume that naming mismatches have been resolved. For instance, nodes labeled by the same value in different trees refer to the same real world concept.

Values in set  $V$  can be grouped to form dimensions. Intuitively, a dimension is a set of semantically related values. For instance, values **Mac**, **Acer** and **Compaq** can be interpreted as values of a dimension **brand**. A semantic interpretation of



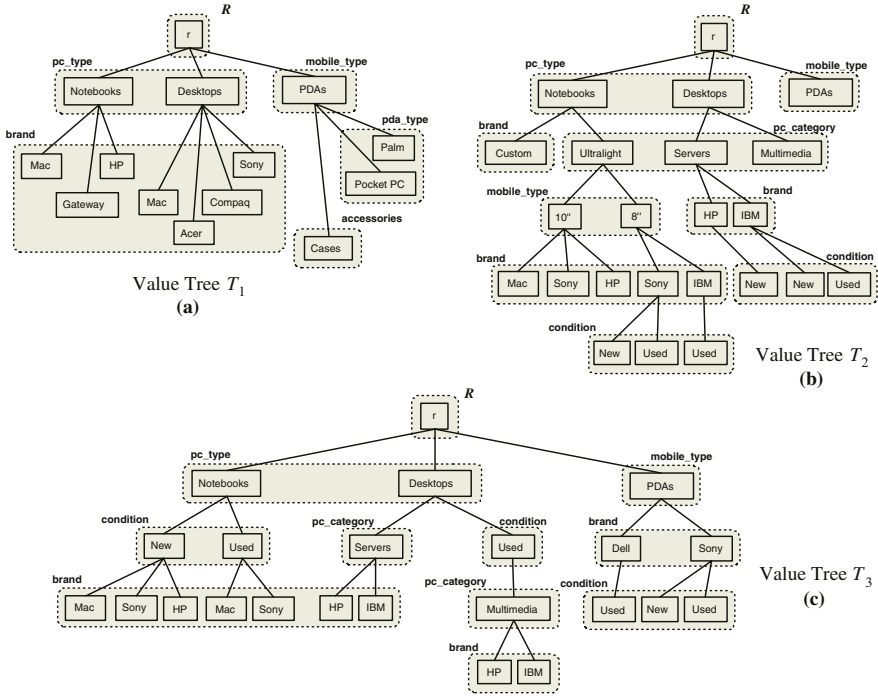


Fig. 1. Value trees  $T_1$ ,  $T_2$  and  $T_3$

values is imposed by a user. A dimension can also be seen as a property with values.

**Definition 2.** Let  $V$  be a set of values that includes a specific value  $r$ . A partition  $\mathcal{D}$  of  $V$  that includes a set  $R$  whose single element is value  $r$ . Each element of  $\mathcal{D}$  is called a dimension.  $\square$

Figure 2 shows a dimension set  $\mathcal{D}$  and the names of its dimensions. We use these dimensions and the value trees of Figure 1 as a running example in this paper.

**Dimension Set**  $D = \{ R, pc\_type, brand, mobile\_type, pda\_type, accessories, pc\_category, condition \}$

**Dimensions:**  
 pc\_type = { Notebooks, Desktops }  
 brand = { Mac, Sony, HP, IBM, Gateway, Acer, Compaq }  
 mobile\_type = { PDAs, 10", 8" }  
 pda\_type = { Palm, Pocket\_PC }  
 accessories = { Cases }  
 pc\_category = { Ultralight, Multimedia, Server }  
 condition = { New, Used }

Fig. 2. A dimension set and its dimensions

A dimension set also partitions the nodes of a value tree. We are interested in value trees where every path from the root to a leaf involves values from distinct dimensions. To describe this type of value trees we introduce the concept of *value trees conforming to a dimension set*.

**Definition 3.** Let  $\mathcal{D}$  be a dimension set over a value set  $V$ . A value tree  $T$  conforms to  $\mathcal{D}$  if and only if there are no two nodes on a path in  $T$  labeled by values that belong to the same dimension in  $\mathcal{D}$ .  $\square$

Consider for example the value trees  $T_1, T_2$  and  $T_3$  of Figure 1. Dotted rectangles labeled by dimensions are used to show the partitioning of nodes into dimensions. The same dimension might label different rectangles in a value tree. In this case, this dimension comprises the nodes confined by all these rectangles. Dimension `pc_type` in  $T_1$  refers to types of personal computers and includes nodes labeled by values `Desktops` and `Notebooks`. Dimension `brand` in  $T_3$  refers to brand names and includes nodes labeled `Mac`, `Sony`, `HP`, `IBM` and `Dell`. All trees  $T_1, T_2$ , and  $T_3$  conform to the dimension set  $\mathcal{D}$  shown in Figure 2.

Nodes labeled by values of the same dimension need not be in the same level of a value tree. For example, in  $T_2$ , the nodes labeled `10"` and `8"` of dimension `mobile_type` are not in the same level as the node labeled `PDAs` of the same dimension. A value of a dimension may not appear in a value tree. For example, the value `Ultralight` of dimension `pc_category` does not appear in value tree  $T_3$  nor in  $T_1$ , although it appears in  $T_2$ . Further, a dimension may have no value in a value tree. For instance, no value of `pc_category` appears in  $T_1$ .

In the following we assume that a dimension set  $\mathcal{D}$  is given and all value trees conform to  $\mathcal{D}$ .

### 3.2 Dimension Graphs

Values of one dimension can label children or descendants of nodes labeled by values of any other dimension in a value tree. However, there are cases where values of one dimension do not label descendants of nodes labeled by values of some other dimension. For example, none of the values `Pocket_PC` and `Palm` of dimension `pda_type` labels a descendant of the nodes labeled by the value `Desktops` or `Notebooks` of dimension `pc_type` in the value tree  $T_1$  of Figure 1. To capture this type of relationship between dimensions in a value tree, we introduce the concept of a dimension graph. Dimension graphs can be automatically extracted from value trees and abstract their structural information. Moreover, they provide semantic query guidance to pose and evaluate queries on value trees (see subsequent sections). Before we give the formal definition of a dimension graph with respect to a value tree, we define dimension graphs as general structures and we present the notion of a dimension precedence.

**Definition 4.** A dimension graph over dimension set  $\mathcal{D}$  is a directed graph whose nodes are dimensions in  $\mathcal{D}$ .  $\square$

A path in a dimension graph is a sequence  $D_1, \dots, D_k$  of distinct nodes such that there is a directed edge from  $D_i$  to  $D_{i+1}$ , where  $1 \leq i \leq k - 1$ .

**Definition 5.** Let  $T$  be a value tree over dimension set  $\mathcal{D}$ . A dimension  $D_i \in \mathcal{D}$  precedes a dimension  $D_j \in \mathcal{D}$  in  $T$  if and only if there are nodes  $n_i$  and  $n_j$  in  $T$  labeled by values  $v_i \in D_i$  and  $v_j \in D_j$ , respectively, such that  $n_j$  is a child node of  $n_i$  in  $T$ .  $\square$

For example, dimension `pc_type` precedes dimension `brand` in  $T_1$  of Figure 1, since a node labeled `Mac` (a value of `brand`) is a child of a node labeled `Desktops` (a value of `pc_type`).

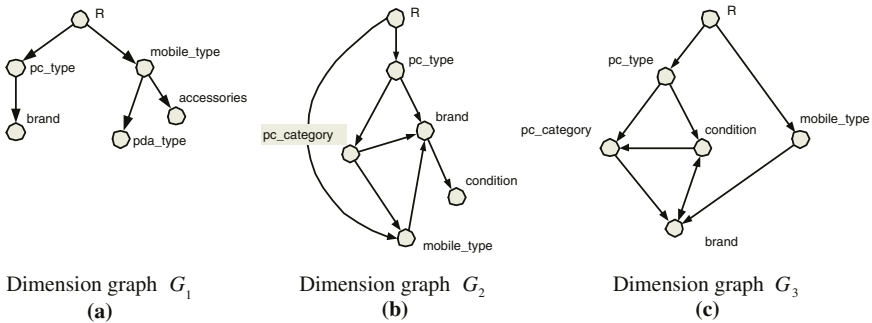
Based on the definitions of dimension graphs as general structures and the notion of dimension precedence, we proceed to define formally dimension graphs with respect to a value tree.

**Definition 6.** Let  $T$  be a value tree over dimension set  $\mathcal{D}$ . A dimension graph  $\mathcal{G}$  of  $T$  is a dimension graph  $(N, E)$ , where  $N$  is a set of nodes and  $E$  is a set of edges defined as follows:

- (a) There is a node  $D$  in  $N$  if and only if there is a value in  $T$  that belongs to dimension  $D$ .
- (b) There is a directed edge in  $E$  from node  $D_i$  to node  $D_j$  if and only if dimension  $D_i$  precedes dimension  $D_j$  in  $T$ .

If  $\mathcal{G}$  is a dimension graph of a value tree  $T$ , we say that  $T$  is a value tree of  $\mathcal{G}$ .  $\square$

Consider for example the value trees  $T_1, T_2$  and  $T_3$  of Figure 1. Figure 3 shows the dimension graphs  $\mathcal{G}_1, \mathcal{G}_2$  and  $\mathcal{G}_3$  of  $T_1, T_2$  and  $T_3$ , respectively. There is an edge from dimension `mobile_type` to dimension `pda_type` in  $\mathcal{G}_1$ , since a node labeled `Palm` (a value of `pda_type`) is a child of a node labeled `PDAs` (a value of `mobile_type`) in value tree  $T_1$ . Looking at the lower left part of value tree  $T_3$ , we note that a node labeled `Mac` (a value of `brand`) is a child of a node labeled `New` (a value of `condition`). However, looking at the lower right part of  $T_3$ , a node labeled `New` is a child of a node labeled `Dell` (another value of dimension `brand`). Thus, dimension `brand` precedes dimension `condition` and vice versa. As a result, there is an edge from dimension `condition` to dimension `brand` and an edge from `brand` to `condition` in  $\mathcal{G}_3$  which are compactly shown in the figures by a double headed edge.



**Fig. 3.** Dimension Graphs

The dimension graph of a value tree has a particular form. The following proposition describes some of its properties.

**Proposition 1.** Consider a dimension graph  $\mathcal{G}$  of a value tree  $T$  over a dimension set  $\mathcal{D}$ . Let  $v_1, \dots, v_k$  be values from the distinct dimensions  $D_1, \dots, D_k \in \mathcal{D}$ , respectively. If  $v_1, \dots, v_k$  label, in that order, nodes on a path in  $T$ , then  $D_1, \dots, D_k$  appear in that order on a path from the root in  $\mathcal{G}$ .  $\square$

## 4 Queries

We present in this section a simple query language and we outline how queries can be evaluated. Our intention is not to provide a full-fledged language. For instance, it does not include selection predicates. Our goal is to show how dimensions can be used to query value trees. Queries in this language are defined on dimension graphs. Roughly speaking, a user poses a query by annotating some dimensions in a dimension graph with permissible sets of values. The answer comprises root-to-leaf paths on the underlying value tree that involve one value from each of these value sets. An interesting feature of the language is that the user has the choice of not specifying or partially specifying parent-child and ancestor-descendant relationships between the annotated dimensions in a query. The system can identify possible orderings of dimensions in the paths of the answer based on the dimension graph only. These orderings are used as patterns for constructing the path expressions that compute the answer of the query on the underlying value tree. All the other orderings of dimensions are excluded from consideration before the computation of the query answer reaches the value tree.

### 4.1 Syntax

A query on a dimension graph comprises annotations of the graph dimensions with sets of values and specifications of precedence relationships between the graph dimensions.

**Definition 7.** Let  $\mathcal{G}$  be a dimension graph over a dimension set  $\mathcal{D}$ . A query  $Q$  on  $\mathcal{G}$  is a pair  $(\mathcal{A}, \mathcal{P})$ , where:

- (a)  $\mathcal{A}$  is a set of expressions of the form  $D_i = A_i$ , where  $D_i$  is a dimension in  $\mathcal{G}$  different than  $R$ , and  $A_i$  is a set of values of dimension  $D_i$  or a question mark (“?”). If  $D_i = A_i$  belongs to  $\mathcal{A}$  we say that  $D_i$  is annotated in  $Q$  and  $A_i$  is called the annotation of  $D_i$  in  $Q$ . Even if not present in  $\mathcal{A}$ , dimension  $R$  is assumed to be an annotated dimension, annotated with the singleton  $\{r\}$ . A dimension can be annotated only once in a query.
- (b)  $\mathcal{P}$  is a set of precedence expressions, which are expressions of the form  $D_i \rightarrow D_j$  or  $D_i \Rightarrow D_j$ , where  $D_i$  and  $D_j$  are annotated dimensions of  $Q$ .

Sets  $\mathcal{A}$  and  $\mathcal{P}$  can be empty.  $\square$

We graphically represent a query  $Q = (\mathcal{A}, \mathcal{P})$  on a dimension graph  $\mathcal{G}$  by labeling its nodes by their annotations in  $\mathcal{A}$  and by adding to it a single (resp. double)

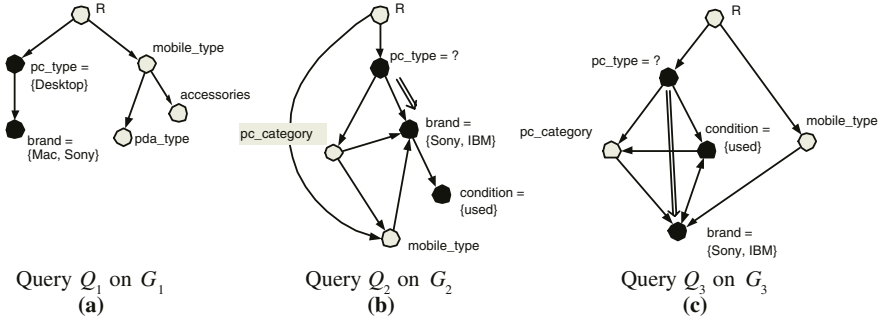


Fig. 4. Graphical Representation of Queries

arrow from node  $D_i$  to node  $D_j$  for every precedence relationship  $D_i \rightarrow D_j$  (resp.  $D_i \Rightarrow D_j$ ) in  $\mathcal{P}$ . Note that arrows are different than directed edges. The unqualified word “arrow” refers indiscreetly to a single or double arrow.

Consider for instance the dimension graphs  $\mathcal{G}_1, \mathcal{G}_2$ , and  $\mathcal{G}_3$  of Figure 3. Figure 4 shows the graphical representation of different queries on these dimension graphs. Annotated nodes are shown in the figures with black circles. Precedence relationships are shown with single or double arrows from one node to another.

Figure 4(a) represents query  $Q_1 = (\mathcal{A}_1, \mathcal{P}_1)$  on dimension graph  $\mathcal{G}_1$ , where  $\mathcal{A}_1 = \{\text{brand} = \{\text{Mac, Sony}\}, \text{pc.type} = \{\text{Desktops}\}\}$  and  $\mathcal{P}_1 = \emptyset$ . In  $Q_1$  we do not specify any precedence relationships between the annotated notes.

In the following we often identify a query with its graphical representation. Figures 4(b) and (c) represent queries  $Q_2$  and  $Q_3$ . A double arrow from node  $\text{pc.type}$  to  $\text{brand}$  denotes the precedence relationship  $\{\text{pc.type} \Rightarrow \text{brand}\}$  in  $Q_2$ .

### 4.2 Semantics

The answer of a query on a value tree  $T$  is a set of root-to-leaf paths in  $T$  compactly represented as a subtree of  $T$ .

**Definition 8.** Let  $\mathcal{G}$  be a dimension graph of a value tree  $T$  over a dimension set  $\mathcal{D}$ , and  $Q$  be a query on  $\mathcal{G}$ . The **answer** of  $Q$  on  $T$  is a subtree  $T'$  of  $T$  such that:

- (a)  $T'$  and  $T$  have the same root  $r$ .
- (b) Every leaf node of  $T'$  is a leaf node of  $T$ .
- (c) Every path from the root to a leaf node in  $T'$  includes one value from every value set annotating a node in  $Q$ .
- (d) Every path from the root to a leaf node in  $T'$  includes one value from every dimension annotated with a question mark in  $Q$ .

Therefore, for every annotated node (with a value set or a question mark) in  $Q$ , there is one value for the corresponding dimension appearing in every path from the root to a leaf node in  $T'$ .

- (e) For every path  $p$  from the root to a leaf node in  $T'$ , and for every precedence relationship  $D_i \rightarrow D_j$  (resp.  $D_i \Rightarrow D_j$ ) in  $Q$ , the value for  $D_j$  is a child (resp. descendent) of the value for  $D_i$  in  $p$ .

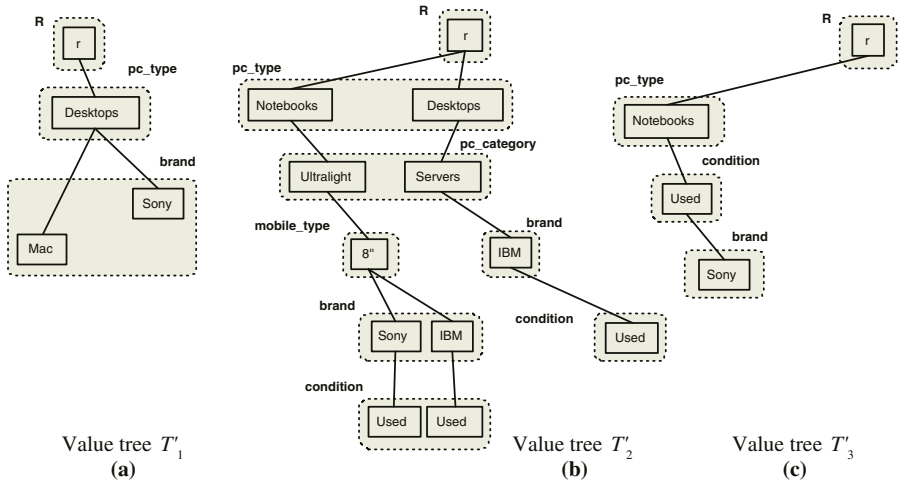


Fig. 5. Query Answers

If there is no such a subtree  $T'$ , we say that the answer of  $Q$  on  $T$  is  $\epsilon, \dots, \epsilon$ . Symbol  $\epsilon$  denotes an empty answer.  $\square$

Annotating a node with a “?” in a query is different than not annotating this node at all. In contrast to a non-annotated node, a node that is annotated with a “?” places a value of the corresponding dimension in every root-to-leaf path in the answer of the query.

Consider the queries  $Q_1, Q_2$  and  $Q_3$  on the dimension graphs  $\mathcal{G}_1, \mathcal{G}_2$ , and  $\mathcal{G}_3$ , respectively, graphically shown in Figure 4. Consider also the value trees  $T_1, T_2$  and  $T_3$  of Figure 1. Figure 5 shows the answers  $T'_1, T'_2$  and  $T'_3$  of  $Q_1, Q_2$  and  $Q_3$  on  $T_1, T_2$  and  $T_3$ , respectively.

Further, consider the query  $Q_4 = (\mathcal{A}_4, \mathcal{P}_4)$ , where:  $\mathcal{A}_4 = \{\text{pc.type} = \{\text{Desktops}\}, \text{brand} = \{\text{HP, Gateway}\}\}$ , and  $\mathcal{P}_3 = \{\text{pc.type} \rightarrow \text{brand}\}$  on the dimension graph  $\mathcal{G}_1$  shown in Figure 3. In the value tree  $T_1$  shown in Figure 1(a) there are values of dimension **brand** that are children of values of dimension **pc.type**. However, there is no root-to-leaf path that involves values **Desktops** and **HP**, or **Desktops** and **Gateway**. Therefore, the answer of  $Q_4$  on  $T_1$  is empty.

### 4.3 Unsatisfiable Queries

A query on a dimension graph  $\mathcal{G}$  is called *unsatisfiable* if its answer is empty on every value tree underlying  $\mathcal{G}$ . Otherwise, it is called *satisfiable*. Detecting the unsatisfiability of a query saves its evaluation on a value tree (which, in any case, produces an empty answer.) In general, this value tree is much larger than its dimension graph which might be needed for detecting the unsatisfiability of the query. The graphical representation of a query provides some intuition on unsatisfiable queries.

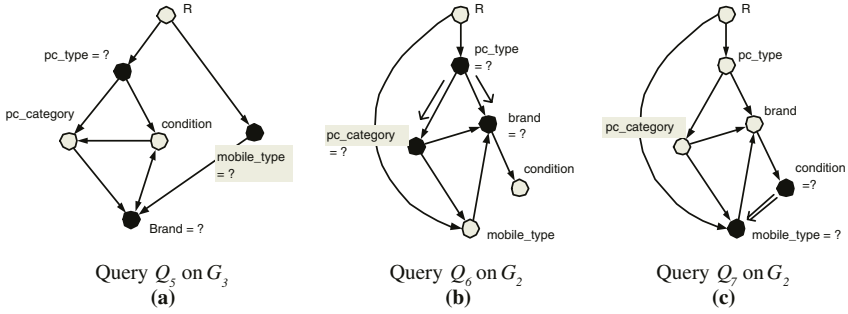


Fig. 6. Unsatisfiable Queries

Consider the dimension graphs  $\mathcal{G}_2$  and  $\mathcal{G}_3$  of Figure 3, and the queries  $Q_5$  on  $\mathcal{G}_3$ , and  $Q_6$  and  $Q_7$  on  $\mathcal{G}_2$  graphically represented in Figure 6. These queries are unsatisfiable.

In query  $Q_5$  of Figure 6(a), there is no path from the root of  $\mathcal{G}_3$  that involves all the annotated nodes. By Proposition 1 there is no root-to-leaf path in a value tree underlying  $\mathcal{G}_3$  that involves values for the annotated dimensions in  $Q_5$ .

In query  $Q_6$  of Figure 6(b), there is a path from the root of  $\mathcal{G}_2$  through all the annotated nodes (e.g. the path  $(R, pc\_type, pc\_category, brand)$ ). However, there are two outgoing single arrows from the same node (node  $pc\_type$ ). Clearly, no two values can be children of the same node in a root-to-leaf path of a value tree underlying  $\mathcal{G}_2$ .

In query  $Q_7$  of Figure 6(c), there is also a path from the root of  $\mathcal{G}_2$  through all the annotated nodes (e.g. the path  $(R, mobile\_type, brand, condition)$ ). However, there is a double arrow from node  $condition$  to node  $mobile\_type$  in  $Q_3$  and no path from node  $condition$  to node  $mobile\_type$  in  $\mathcal{G}_2$ . By Proposition 1 there is no root-to-leaf path in a value tree underlying  $\mathcal{G}_2$  that involves a value for dimension  $condition$  preceding a value for dimension  $mobile\_type$ .

More generally, we can show the following result that provides sufficient conditions for a query to be unsatisfiable.

**Proposition 2.** A query  $Q$  on a dimension graph  $\mathcal{G}$  is unsatisfiable if one of the following conditions holds:

- (a) Arrows in  $Q$  form a directed cycle.
- (b) There are precedence relationships  $D \rightarrow D_i$  and  $D \rightarrow D_j$  or precedence relationships  $D_i \rightarrow D$  and  $D_j \rightarrow D$  in  $Q$  ( $D_i \neq D_j$ ).
- (c) There is a precedence relationship  $D_i \rightarrow D_j$  in  $Q$  but no edge from node  $D_i$  to node  $D_j$  in  $\mathcal{G}$ .
- (d) There is a precedence relationship  $D_i \Rightarrow D_j$  in  $Q$  but no edge from node  $D_i$  to node  $D_j$  in the dimension graph  $\mathcal{G}$  (in other words, no path from node  $D_i$  to node  $D_j$  in  $\mathcal{G}$ ).
- (e) The annotated nodes in  $Q$  are not on a path from the root of  $\mathcal{G}$ . □

In order to provide necessary conditions for query unsatisfiability, we introduce the concept of an answer path of a query.

**Definition 9.** Let  $Q$  be a query on a dimension graph  $\mathcal{G}$ . An answer path  $p$  of  $Q$  in  $\mathcal{G}$  is a path  $p$  in  $\mathcal{G}$  from the root of  $\mathcal{G}$  such that:

- (a) All the annotated dimensions in  $Q$  are on  $p$ , and  $p$  ends on an annotated dimension of  $Q$ .
- (b) If there is a precedence relationship  $D_i \rightarrow D_j$  (resp.  $D_i \Rightarrow D_j$ ) in  $Q$ , then  $D_j$  is a child (resp. descendent) of  $D_i$  in  $p$ . □

Consider for instance the query  $Q_2$  on dimension graph  $\mathcal{G}_2$  and the query  $Q_3$  on dimension graph  $\mathcal{G}_3$ , which are shown in Figures 4(b) and 4(c), respectively. One can identify the following answer paths for query  $Q_2$  in  $\mathcal{G}_2$ :

- $R, pc\_type, brand, condition$
- $R, pc\_type, pc\_category, brand, condition$
- $R, pc\_type, pc\_category, mobile\_type, brand, condition$

The answer paths for query  $Q_3$  in  $\mathcal{G}_3$  are:

- $R, pc\_type, condition, brand$
- $R, pc\_type, pc\_category, brand, condition$

The following proposition provides necessary and sufficient conditions for a query to be unsatisfiable.

**Proposition 3.** A query  $Q$  on a dimension graph  $\mathcal{G}$  is unsatisfiable if and only if there is no answer path of  $Q$  in  $\mathcal{G}$ . □

### 4.4 Query Evaluation

When evaluating a query, we first check it for satisfiability. If a query is satisfiable, we proceed to compute its answer on a value tree in three steps. In the first step, we compute all the answer paths of the query. In the second step, we generate path expressions based on the answer paths. In the third step we evaluate the path expressions on the value tree and compose the answer of the query.

To represent path expressions, we use a notation similar to that of XPath [4]. The fragment of XPath we use involves node names ( $v_i$ ), child axis ( $/$ ), descendant axis ( $//$ ), wildcards ( $*$ ), unions ( $|$ ). The expression  $(v_1|...|v_m)$  represents any node name in the set  $\{v_1, \dots, v_m\}$ . For a dimension  $D$ , we use the expression  $*_D$  as an abbreviation for the expression  $(v_1|...|v_n)$ , where  $\{v_1, \dots, v_n\} = D$ .

Given an answer path, we construct a corresponding path expression as follows. Let  $R, D_1, \dots, D_k$  be an answer path of a query  $Q$ . The corresponding path expression has the form  $r/\theta_1/\dots/\theta_k$ , where, for  $i = 1, \dots, k$ ,

$$\theta_i = \begin{cases} (v_1|...|v_m) & \text{if } D_i \text{ is annotated with the value set } \{v_1, \dots, v_m\} \\ *_D & \text{if } D_i \text{ is annotated with a "?" or if } D_i \text{ is not annotated} \end{cases}$$

Notice that even though nodes annotated with a “?” are treated the same way as non-annotated ones in the construction of path expressions for a query, they affect differently the answer of a query since they are taken into account in the identification of answer paths for that query.



Before showing what the result of a path expression on a value tree is, we introduce the concept of a merge of a set of value trees (or paths). Let  $T_1, \dots, T_k$  be a set of value trees having the same root  $r$ . The *merge* of  $T_1, \dots, T_k$ , denoted  $T_1 \cup \dots \cup T_k$ , is a minimal<sup>1</sup> value tree which has  $T_1, \dots, T_k$  as subtrees. It is not difficult to see that this value tree is unique.

We show now what is the result of a path expression on a value tree. Let  $e$  be a path expression and  $T$  be a value tree. Let also  $P$  be the set of paths from the root of  $T$  to the leafs of  $T$  that satisfy  $e$ . The result  $res(e, T)$  of a path expression  $e$  on a value tree  $T$  is the value tree  $\bigcup_{p \in P} p$ . Note that the result of a path expression is different than the result of the same XPath expression. The result of a path expression is a value tree while the result of the same XPath expression is a set of nodes [4]. We can use XQuery [5] to compute the result of a path expression as it is defined here.

The answer of a query on a value tree can be computed by merging the results of its path expressions on the value tree. Let  $E = \{e_1, \dots, e_n\}$  be the set of path expressions constructed from all the answer paths of a query  $Q$ . The answer of  $Q$  on a value tree  $T$  is the value tree  $\bigcup_{i \in [1, n]} res(e_i, T)$ .

As an example consider the query  $Q_2$  on dimension graph  $\mathcal{G}_2$ , which is shown in Figure 4(b). The answer paths for  $Q_2$  in  $\mathcal{G}_2$  are shown in Section 4.3. These answer paths generate the following path expressions:

```
r/*pc_type/(Sony|IBM)/Used
r/*pc_type/*pc_category/(Sony|IBM)/Used
r/*pc_type/*pc_category/*mobile_type/(Sony|IBM)/Used
```

Evaluating these path expressions on the value tree  $T_2$  of Figure 1(b), one can see that the result of the first path expression is an empty value tree. In contrast, the second path expression contributes one path, while the third one contributes two paths to the answer of  $Q_2$  on  $T_2$  (Figure 5(b)).

Consider also the query  $Q_3$  on dimension graph  $\mathcal{G}_3$ , which is shown in Figure 4(c). The answer paths for  $Q_3$  in  $\mathcal{G}_3$  are shown in Section 4.3. They generate the following path expressions:

```
r/*pc_type/Used/(Sony|IBM)
r/*pc_type/*pc_category/(Sony|IBM)/Used
```

Of those path expressions, evaluating the second one on the value tree  $T_3$  of Figure 1(c) results in an empty value tree. Only the first one contributes a path to the answer of  $Q_3$  on  $T_3$  (Figure 5(c)).

## 4.5 Querying Multiple Value Trees

Consider the value trees  $T_1, \dots, T_n$  over a dimension set  $\mathcal{D}$  and let  $\mathcal{G}_1, \dots, \mathcal{G}_n$  be their dimension graphs respectively. The dimension graphs  $\mathcal{G}_1, \dots, \mathcal{G}_n$  are not necessarily the same. In order to query the value trees  $T_1, \dots, T_n$  together, we need a “global” dimension graph. Such a graph  $\mathcal{G}$  can be constructed by merging

<sup>1</sup> Minimality is meant with respect to the number of nodes or edges.

the dimension graphs  $\mathcal{G}_1, \dots, \mathcal{G}_n$ . A query  $Q$  on  $\mathcal{G}$  is defined on a dimension graph  $\mathcal{G}_i$  if it does not involve dimensions that occur in  $\mathcal{G}$  but not in  $\mathcal{G}_i$ . Otherwise, it is not defined on  $\mathcal{G}_i$  and it returns no answers. If  $Q$  is defined on  $\mathcal{G}_i$ , it can be checked for consistency and evaluated as described in the previous sections. Notice that a query on the global dimension graph  $\mathcal{G}$  can be applied to any of the  $\mathcal{G}_i$ s without the use of mapping rules.

## 5 Conclusion

We presented a method for querying tree structures, called value trees. Our approach exploits semantic information for the nodes of value trees. A semantic relationship between nodes in value trees was captured by the concept of a dimension. Dimension graphs were defined to capture structural information on the dimensions of a value tree. However, dimension graphs are not plain structural summaries of value trees, but rather semantically richer constructs that assist query evaluation. We designed a query language to query value trees. Queries are specified on the dimensions of the value tree and can optionally involve parent-child and ancestor-descendant relationships between these dimensions. A query is not restricted by the structure of a specific value tree. We provided necessary and sufficient conditions for query unsatisfiability and we presented a technique for evaluating satisfiable queries. Finally, we showed how dimension graphs can be used to query multiple value trees in the presence of structural differences and inconsistencies.

We are currently working towards two directions. We are first elaborating on how our framework can be used for integrating tree structured data. In particular, we examine how to apply our techniques to query and integrate XML data sources that conform to different DTDs. The second research direction involves extending our query language with additional features, e.g. branching path expressions and disjunctions.

## References

1. Exchangeable Faceted Metadata Language, (XFML), 2003, <http://www.xfml.org/>.
2. XML Topic Maps (XTM), 2001, <http://www.topicmaps.org>.
3. World Wide Web Consortium site (W3C), <http://www.w3c.org>.
4. XML Path Language (XPath). World Wide Web Consortium site. W3C, 2003-2005, <http://www.w3c.org/TR/xpath20/>.
5. XML Query (XQuery). World Wide Web Consortium site, The Architecture Domain. W3C, 2003-2005, <http://www.w3.org/XML/Query>.
6. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.
7. B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-based Integration of XML Web Resources. In *Proc. of the ICSW'02 Conference, Sardinia, Italy, 2002*.
8. R. Behrens. A Grammar-based Model for XML Schema Integration. In *Proc. of the BNCOD'00 Conference, Exeter, UK, 2000*.

9. S. Bergamaschi, F. Guerra, and M. Vincini. A Data Integration Framework for E-commerce Product Classification. In *Proc. of the ICSW'02 Conference, Sardinia, Italy*, 2002.
10. P. Buneman, S. B. Davidson, M. F. Fernandez, and D. Suciu. Adding Structure to Unstructured Data. In *Proc. of the ICDT'97 Conference, Delphi, Greece, 1997*.
11. A. B. Chaudhri, A. Rashid, and R. Zicari. *XML Data Management*. Addison Wesley, 2003.
12. V. Christophides, S. Cluet, and J. Simeon. On Wrapping Query Languages and Efficient XML Integration. In *Proc. of the ACM SIGMOD'00 Conference, USA, 2000*.
13. S. Cluet, P. Veltri, and D. Vodislav. Views in a Large Scale XML Repository. In *Proc. of the VLDB'01 Conference, Rome, Italy, 2001*.
14. M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. In *Proc. of the ACM SIGMOD'00 Conference, Dallas, Texas, USA, 2000*.
15. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. of the VLDB'97 Conference, Athens, Greece, 1997*.
16. D. Kim, J. Kim, and S.-G. Lee. Catalog Integration for Electronic Commerce through Category-hierarchy Merging Technique. In *Proc. of the RIDE'02 Workshop, San Jose, USA, 2002*.
17. M. L. Lee, L. H. Yang, W. Hsu, and X. Yang. Xclust: Clustering XML Schemas for Effective Integration. In *Proc. of the CIKM'02 Conference, Virginia, USA, 2002*.
18. I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries over Heterogeneous Data Sources. In *Proc. of the VLDB'01 Conference, Rome, Italy, 2001*.
19. P. J. Marron, G. Lausen, and M. Weber. Catalog Integration Made Easy. In *Proc. of the ICDE'03 Conference, Bangalore, India (poster), 2003*.
20. E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4), 2001.
21. Y. Tzitzikas, N. Spyratos, P. Constantopoulos, and A. Analyti. Extended Faceted Taxonomies for Web Catalogs. In *Proc. of the WISE'02 Conference, Singapore, Dec 2002*.

# Workflow Resource Patterns: Identification, Representation and Tool Support\*

Nick Russell<sup>1</sup>, Wil M.P. van der Aalst<sup>2,1</sup>, Arthur H.M. ter Hofstede<sup>1</sup>,  
and David Edmond<sup>1</sup>

<sup>1</sup> Centre for IT Innovation, Queensland University of Technology,  
P.O. Box 2434, Brisbane QLD 4001, Australia

{n.russell, a.terhofstede, d.edmond}@qut.edu.au

<sup>2</sup> Department of Technology Management, Eindhoven University of Technology,  
P.O. Box 513, NIL-5600 MB, Eindhoven, The Netherlands

w.m.p.v.d.aalst@tm.tue.nl

**Abstract.** In the main, the attention of workflow researchers and workflow developers has focussed on the process perspective, i.e., control-flow. As a result, issues associated with the resource perspective, i.e., the people and machines actually doing the work, have been largely neglected. Although the process perspective is of most significance, appropriate consideration of the resource perspective is essential for successful implementation of workflow technology. Previous work has identified recurring, generic constructs in the control-flow and data perspectives, and presented them in the form of control-flow and data patterns. The next logical step is to describe workflow resource patterns that capture the various ways in which resources are represented and utilised in workflows. These patterns include a number of distinct groupings such as push patterns (“the system pushes work to a worker”) and pull patterns (“the worker pulls work from the system”) to describe the many ways in which work can be distributed. By delineating these patterns in a form that is independent of specific workflow technologies and modelling languages, we are able to provide a comprehensive treatment of the resource perspective and we subsequently use these patterns as the basis for a detailed comparison of a number of commercially available workflow management systems.

**Keywords:** Workflow, Patterns, Resources, Business Process Modelling.

## 1 Introduction

Over the last decade there has been increasing interest in *Workflow Management Systems* (PAIS), i.e., systems that are used to support, control, and/or

---

\* This work was partially supported by the Dutch research school BETA as part of the *PATINT* program and the Australian Research Council under the Discovery Grant *Expressiveness Comparison and Interchange Facilitation between Business Process Execution Languages*.

monitor business processes. Typical examples of systems that are driven by implicit or explicit process models are WorkFlow Management (WFM) systems, Enterprise Resource Planning (ERP) systems and Customer Relationship Management (CRM) systems. These systems can be configured to support specific business processes. Recently, several languages have been proposed to support process-orientation in the context of web-services (cf. BPEL4WS, BPML, WSCI, etc.). The support of IBM, Microsoft, HP and SAP for a language like BPEL4WS (Business Process Execution Language for Web Services, [6]) reinforces the fact that process-awareness has become one of the cornerstones of information systems development. Existing languages and tools focus on control-flow and combine this focus with mature support for data in the form of XML and database technology. As a result, control-flow and data-flow are well-addressed by existing languages and systems. Unfortunately, [www.workflowpatterns.com](http://www.workflowpatterns.com) continues to be the case even with relatively recent advances such as the language BPEL4WS [6] which does not provide any degree of direct support for resources in business processes based on web-services. Similarly, languages like XPDL [13], the “Lingua Franca” proposed by the Workflow Management Coalition (WfMC), has a very simplistic view of the resource perspective and provides minimal support for modelling workers, organisations, work distribution mechanisms, etc. A quote attributable to John Seely Brown (a former Chief Scientist at Xerox) succinctly captures the current predicament: “Processes don’t do work, people do!”. In other words, it is not sufficient to simply focus on control-flow and data issues when capturing business processes, the resources that enable them need to be considered as well.

In this paper, we focus on the resource perspective. The resource perspective centres on the modelling of resources and their interaction with a PAIS. Resources can be human (e.g., a worker) or non-human (e.g., plant and equipment), although our focus will be on human resources. Although PAIS typically identify human resources, they know very little about them. For example, in a workflow system like Staffware a human resource is completely specified by the work queue (s)he can see. This does not do justice to the capabilities of the people using such systems. Staffware also does not leave a lot of “elbow room” for its users since the only thing they can do is execute the work items in their work queues, i.e., people are treated as automatons and have little influence over the way in which work is distributed. The limitations of existing systems triggered the research presented in this paper. By identifying resource patterns and providing a critical analysis of existing workflow management systems we hope to encourage workflow researchers and workflow developers to improve the resource perspective in future offerings.

This work extends the [www.workflowpatterns.com](http://www.workflowpatterns.com)<sup>1</sup> to the resource perspective. This research project seeks to identify recurring generic workflow constructs through review of the conceptual foundations of workflow systems together with

---

<sup>1</sup> See [www.workflowpatterns.com](http://www.workflowpatterns.com) for more information, i.e., animations, papers, tool evaluations, etc.

a comprehensive survey of commercial product offerings. These constructs are then described in the form of patterns. Initially, it examined control-flow dependencies in workflow languages [2]. Later it was extended to include web-services composition languages [1] and the data perspective [11]. This work<sup>3</sup> has directly influenced tool selection processes, commercial and open-source workflow systems, and workflow standards. In this paper, we adopt an approach similar to that in [2] although in this case, the focus is on the resource perspective.

The remainder of the paper is organised as follows. We first introduce some basic concepts relating to workflow management in general and resource modelling in particular. Then we describe a series of selected resource patterns drawn from the various categories that have been identified. Note that of the more than 40 resource patterns identified, we only describe a few typical examples and refer the reader to [12] for a comprehensive discussion of the complete set of patterns. Section 4 evaluates five existing workflow products using these patterns. Section 5 discusses related work and Section 6 concludes the paper.

## 2 Workflow and Resource Concepts

Before we describe the resource patterns in detail, we present a standard set of definitions for the various workflow concepts that we will utilise throughout this paper. In doing so, we first introduce some general terminology for workflow models and then focus on the concepts relating to the resource perspective.

### 2.1 Workflow Model

A *workflow model* or *business process model* is a description of a business process in sufficient detail that it is able to be directly executed by a *workflow engine*. A *workflow model* is composed of a number of *tasks* which are connected in the form of a directed graph. An executing instance of a workflow model is called a *workflow instance* or *business process instance*. There may be multiple cases of a particular workflow model running simultaneously, however each of these is assumed to have an independent existence and they typically execute without reference to each other.

There is usually a unique first task and a unique final task in a workflow. These are the tasks that are first to run and last to run in a given workflow case.

A *task* corresponds to a single unit of work. Four distinct types of task are denoted: *simple task*, *block task*, *sub-workflow task* and *parallel task*. An *simple task* is one which has a simple, self-contained definition (i.e. one that is not described in terms of other workflow tasks) and only one instance of the task executes when it is initiated. A *block task* is a complex action which has its implementation described in terms of a *sub-workflow*. When a *block task* is started, it passes control to the first task(s) in its corresponding *sub-workflow*. This *sub-workflow* executes to completion and at its conclusion, it passes control back to the *block task*. E.g. block task C is defined in terms of the sub-workflow comprising tasks, X, Y and Z. A *parallel task* is a task that may have multiple

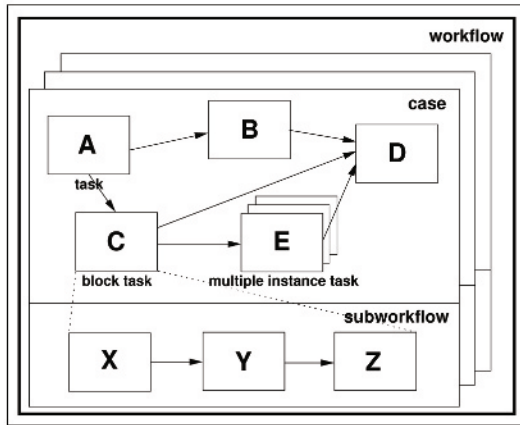


Fig. 1. A overview of the basic workflow concepts

distinct execution instances running concurrently within the same workflow case and a *block task* is a combination of the two previous constructs and denotes a task that may have multiple distinct execution instances each of which is block structured in nature (i.e. has a corresponding *block task*).

The control flow between tasks occurs via the *control flow*, which is indicated by a solid arrow between tasks.

Each invocation of a task that executes is termed a *work item*. Usually there is one work item initiated for each task in a given case however for a *multiple instance task*, there may be several associated work items that are created when the task is initiated. Similarly, where a task forms part of a loop, a distinct work item is created for each iteration.

In general a work item is directed to a resource for execution (although a resource is not required to undertake automatic tasks). There are a variety of ways by which this may be achieved which will be discussed subsequently.

A task may initiate one or several tasks when it completes (i.e. when a work item corresponding to it completes). This is illustrated by an arrow from the completing task to the task being initiated e.g. in Figure 1, task B is initiated when task A completes. This may also occur conditionally and where this is the case, the edge between tasks indicates the condition that must be satisfied for the subsequent task to be started.

Figure 1 shows the control-flow perspective and only hints at the other perspectives (e.g., resources and data). Nevertheless, it sets the scene for discussing the resource perspective.

## 2.2 Resource Perspective

Typically, resources are needed to execute *work items*, i.e., invocations of tasks for specific cases. A resource is an entity that is capable of doing work and can be classified as either *dedicated* or *shared*, i.e., a resource that does not

correspond to an actual person - e.g. plant and equipment. As mentioned in the introduction, we focus on human resources. However, many of the concepts and patterns also apply to non-human resources.

A human resource is typically a member of an organisation. An organisation is a formal grouping of resources that undertake work items pertaining to a common set of business objectives. They usually have a specific location within that organisation, which may have specific responsibilities associated with it. They may also be a member of one or more departments which are permanent groups of human resources within the organisation that undertake work items relating to a specific set of business functions. Details of the organisational structure in which a human resource may operate are not discussed in this paper. Interested readers are referred to [12], where these issues are examined in more detail.

A resource may have one or more associated roles which serve as another grouping mechanisms for human resources with similar job roles or responsibility levels e.g. managers, union delegates etc. Individual resources may also possess skills or attributes that further clarify their suitability for various kinds of work items. These may include qualifications and skills as well as other job-related or personal attributes such as specific responsibilities held or previous work experience. They may also have characteristics which further describe specific characteristics that they may possess that could be of interest when allocating work items. A comprehensive resource meta-model is presented in [12].

### 2.3 Lifecycle of a Work Item

Of particular interest from a resource perspective is the manner in which work items are advertised and ultimately bound to specific resources for execution. Figure 2 illustrates the lifecycle of a work item in the form of a state transition diagram from the time that a work item is created through to final completion or failure. It can be seen that there are a series of potential states that comprise this process.

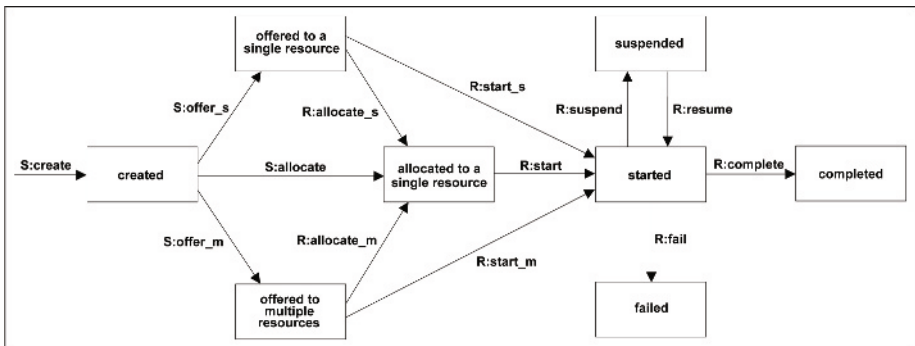


Fig. 2. State Transition Diagram for Work Distribution



Each node in Figure 2 represents a possible state of a work item. Each edge within this diagram is prefixed with either an `S` or an `R` indicating that the transition is initiated by the workflow system (S) or resource (R) respectively.

Initially a work item comes into existence in the `WIP` state. This indicates that the preconditions required for its enablement have been satisfied and it is capable of being executed. At this point however, the work item has not been allocated to a resource for execution. In state `WIP` the system can take one of three possible courses of action: (1) offer the work item to a single resource, (2) allocate it to a resource, or (3) offer it to multiple resources. The difference between `WIP` and `OFFERED` to a single resource is subtle. If a work item is allocated, there is a commitment on the part of the resource to execute the work item. Note that this commitment may be imposed by the system via `WIP` or by the resource him/herself (via `OFFERED` or `ALLOCATED`). No such commitment is implied for offered work items. Depending on the system and the resource, an offered or allocated work item can be started (cf., state `STARTED`). Started work items can be completed, suspended (followed by a resume) or failed.

It is important to note that Figure 2 is just an example of a lifecycle model for work items. Some of the states and transitions may not be present in a given system. Moreover, when we consider more advanced patterns we need to assume an extended lifecycle model, e.g., to capture delegation.

### 3 Resource Patterns

The patterns presented in this section are intended to be language independent and do not assume a concrete syntax. In the absence of a commonly agreed workflow model, the aim is to define them in a form that ensures they are applicable to the broadest possible range of PAIS. The patterns are grouped into a number of categories: `Creation Patterns`, `Resource Allocation Patterns`, `Resource Management Patterns`, `Resource Delegation Patterns`, `Resource Scheduling Patterns`, `Resource Monitoring Patterns`, and `Resource Termination Patterns`. In this section we only describe a selection of the resource patterns. For a complete overview and detailed descriptions we refer the reader to [12].

#### 3.1 Creation Patterns

The first category of patterns refers to the various restrictions that can be defined for a work item at design time. These restrictions can refer to the range of resources that may execute the work item (“Who?”), the time at which the work item should be executed (“When?”) and the location at which the work item can be processed (“Where?”). Note that the creation patterns refer to the design phase, i.e., before there are any work items in existence. Therefore, they cannot be linked directly to Figure 2.

We have identified eleven creation patterns but in this paper we only describe the second of these: role-based allocation.

**Pattern 2: R-RBA (Role-Based Allocation)**

*Description* The ability to specify at design time that a task can only be executed by resources which correspond to a given role.

*Example* Instances of the `Task` must be executed by a `Resource`.

*Motivation* Perhaps the most common approach to work item allocation within workflow systems, role-based allocation offers the means for the workflow engine to route work items to suitably qualified resources at runtime. The decision as to which resource actually receives a given work item is deferred until the moment at which it becomes runnable and requires a resource allocation in order for it to proceed. The advantage offered by role-based allocation is that roles can be defined for a given workflow process that define the various classes of available resources to undertake work items. Task definitions within the process model can nominate the specific role to which they should be routed; however the actual population of individual roles does not need to occur until runtime.

*Implementation* All of the workflow systems examined support role-based allocation.

The other ten creation patterns describe additional constraints that can be specified at design time, e.g., the “separation of duties” pattern (also known as the four-eyes principle) which specifies that certain tasks cannot be executed by the same person within the same case. Table 1 lists all eleven creation patterns.

**3.2 Push Patterns**

Push patterns refer to the ability of the `Task` to offer or allocate work items. These patterns relate to state transitions `S:offer_s`, `S:offer_m` and `S:allocate` shown in Figure 2. Transition `S:offer_s` corresponds to a work item being offered to a single resource, `S:offer_m` corresponds to a work item being offered to multiple resources, and `S:allocate` corresponds to a work item being directly allocated to a specific resource immediately after it has been created. The main difference between an `offer_s` work item and an `offer_m` work item is the level of commitment implied (i.e. whether the resource is committed to executing the work item or has merely been advised of its existence). In the case where a work item is offered to multiple resources, the initiative is left with the resources to determine which of them will actually execute it. This contrasts with allocated work items where the decision is made by the system. However, if the work item is offered to a single resource, the difference is more subtle and in some systems it will be difficult, if not impossible, to distinguish between `S:offer_s` and `S:allocate_s`.

There are nine push patterns (cf. Table 1). Here we only discuss two of them: patterns 13 and 17.

**Pattern 13: R-DBOM (Distribution by Offer – Multiple Resources)**

*Description* The ability to offer a work item to a group of selected resources.

*Example* The `Task` work item is offered to multiple `Resource`.

**Motivation** Offering a work item to multiple resources is the workflow analogy to the act of “calling for a volunteer” in real life. It provides a means of advising a suitably qualified group of resources that a work item exists but leaves the onus with them as to who actually commits to undertake the activity. Note that this pattern corresponds to the ability to take transition `S:offer_m` in Figure 2.

**Implementation** Several workflow engines support the notion of work groups and allow work items to be allocated to them. A work group is a group of resources with a common organisational focus. When a work item is allocated to the group, each of the group members is advised of its existence, but until one of them commits to starting it and advises the workflow engine of this fact, it remains on the work queue for each of the resources.

There are several ways in which a resource can be advised of group work items: (a) it may appear on each of their individual work queues, (b) each resource may have a distinct work queue for group items on which it may appear, or (c) all resources in a work group may have the ability to view a shared group work queue in addition to their own dedicated work queue.

Different workflow engines handle the offering of a work item to multiple resources in different ways:

- WebSphere treats work items offered to multiple resources in the same way as work items allocated to a specific resource and they appear on the work list of all resources to whom they are offered. When an advertised work item is accepted by one of the resources to which it is offered, it is removed from the work lists of all other resources.
- Staffware and COSA support the concept of distinct user specific work queues and group work queues. Where an advertised work item is accepted by a resource, it remains on the group work list but is not able to be selected for execution by other resources.
- iPlanet supports distinct work queues for offered and queued (i.e. allocated) work items. Once an advertised work item has been accepted by a resource, it is removed from all offered work queues and only appears on the work queue for the resource which has accepted it.

### Pattern 17: R-SHQ (Shortest Queue)

**Description** The ability to allocate a work item to the resource that has the least number of work items allocated to it.

**Example** The `allocate` transition is allocated to the `resource` who has the least number of operations allocated to them.

**Motivation** This allocation mechanism seeks to expedite the throughput of a workflow process by ensuring that work items are allocated to the resource that is able to undertake them in the shortest possible timeframe. Typically the shortest timeframe means the resource with the shortest work queue although other interpretations are possible. Note that this pattern corresponds to a specific realization of transition `S:allocate` in Figure 2, i.e., it is an allocation by the system based on the number of already allocated work items.

**Implementation** In order to implement this allocation method, workflow engines need to maintain information on the work items currently allocated to resources and make this information available to the work item distribution algorithm. COSA directly implements this allocation method via the `fewwork()` function. iPlanet provide facilities for programmatically extending the work item distribution algorithm indirectly enabling this pattern to be achieved.

### 3.3 Pull Patterns

While push patterns focus on work distribution from the system's perspective, pull patterns consider the issue from the resource's viewpoint. After the system offers or allocates a work item, the resource can take the initiative to allocate or start a work item. This is reflected by the transitions `R:allocate_s`, `R:allocate_m`, `R:start_s`, `R:start_m`, and `R:start` in Figure 2. There are six pull patterns, all related to these transitions. Here, we only discuss two of them: patterns 21 and 23.

#### Pattern 21: R-RIA (Resource-Initiated Allocation)

**Description** The ability for a resource to commit to undertake a work item without needing to commence working on it immediately.

**Example** The `allocate` selects the `workitem` work items that she will undertake today although she only commences working on one of these now.

**Motivation** This pattern provides a means for a resource to signal its intention to execute a given work item at some point in time although it may not commence working on it immediately. As a consequence, the work item is considered to be allocated to the resource and it cannot be allocated to or executed by another resource. Clearly this pattern corresponds to transitions `R:allocate_s` and `R:allocate_m`.

**Implementation** The implementation of this pattern generally involves the removal of the work item from a globally accessible or shared work list and its placement on a work queue specific to the resource to which it is allocated. FLOWer directly supports this pattern through its case query construct. COSA allows a resource to reserve a work item that is displayed on a shared or global worklist for later execution by a user; however in doing so, the entire process instance is locked by the resource until the work item is completed.

#### Pattern 23: R-RIEO (Resource-Initiated Execution – Offered Work Item)

**Description** The ability for a resource to select a work item offered to it and commence work on it immediately.

**Example** The `start` selects the next `workitem` work item from those offered and commences work on it.

**Motivation** In some cases it is preferable to view a resource as being committed to undertaking a work item only when the resource has actually started working on it, i.e., there is no explicit allocation phase. This approach to work distribution

effectively speeds throughput by eliminating the notion of work item allocation. Work items remain on offer to the widest range of appropriate resources until one of them actually indicates they can commence work on it. Only at this time is the work item removed from being on offer and allocated to a specific resource. This pattern corresponds to transitions `R:start_s` and `R:start_m` in Figure 2.

**Implementation** This approach to work distribution is adopted by Staffware, WebSphere and COSA for shared work queues (e.g. group queues). For these systems, a work item remains on the queue until a resource indicates that it has commenced it. At this point, its status changes and no other resource can execute it although it remains on the shared queue until it is completed. iPlanet adopts this approach for all work items and effectively presents each resource with a single amalgamated queue of work items allocated directly to it as well as those offered to a range of resources. The resource must indicate when it wishes to commence a work item. This results in the status of the work item changing and it being removed from any other work queues on which it might have existed. Patterns 21 and 23 do not convey the fact that work can be offered in various ways, e.g. the order in which work is offered may be determined by the system or by the resources. Table 1 provides a complete listing of pull patterns and gives an insight into the various bases on which work items can be offered to resources.

### 3.4 Detour Patterns

The state transitions shown in Figure 2 refer to the basic lifecycle of a work item, i.e., the “normal way” of handling work. However, there are situations where pre-existing work allocations are interrupted either by the workflow system or at the instigation of the associated resource. As a consequence of such events, the normal sequence of state transitions for a workflow item is varied. Patterns for dealing with these exceptional situations are called detour patterns. The range of possible scenarios for detour patterns is illustrated in Figure 3.

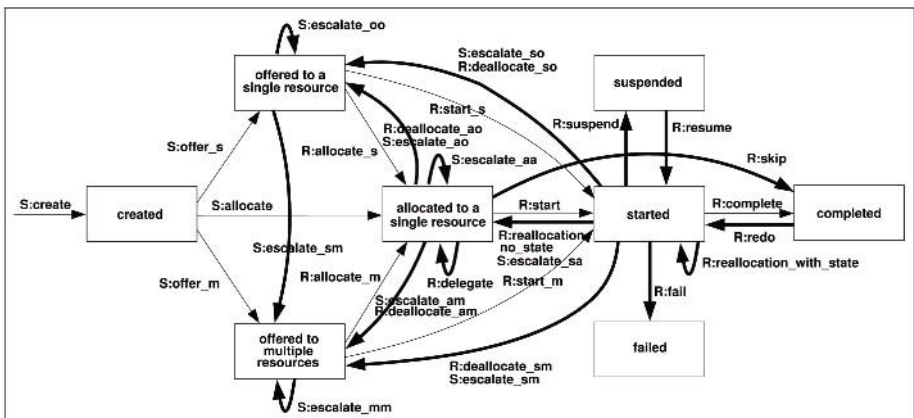


Fig. 3. Detour Patterns

As shown in Figure 3, detour patterns allow for alternative state transitions. Possible detour patterns include:

- `allocate` - where a resource allocates a work item previously allocated to it to another resource.
- `reallocate` - where the workflow system attempts to progress a work item that has stalled by offering or allocating it to another resource.
- `release` - where the system makes a previously allocated or started work item available for offer and subsequent allocation.
- `transfer` - where a resource allocates a work item that it has started work on to another resource.
- `suspend`, `resume`, `reschedule` - where a resource temporarily suspends execution of a work item and recommences execution of it at a later time.
- `skip` - where a resource elects to skip the execution of a work item allocated to it.
- `repeat` - where a resource repeats execution of a work item completed earlier.
- `advance` - where a resource executes a work item that is ahead of the current execution point of a workflow case.

In this paper we only describe one of the detour patterns in detail.

**Pattern 27: R-D (Delegation)**

*Description* The ability for a resource to allocate a work item previously allocated to it to another resource.

*Example* Before going on leave, the `John Smith` passed all of his outstanding work items onto the `John Doe`.

*Motivation* Delegation provides a resource with a means of re-routing work items that it is unable to execute. This may be because the resource is unavailable (e.g. on vacation) or because they do not wish to take on any more work. The pattern corresponds to transition `R:delegate` in Figure 3.

*Implementation* Generally the ability to delegate a work item is included in the client work list handler for a workflow engine. Staffware, WebSphere and COSA allow individual queued work items to be redirected to a given resource. COSA also has an enhanced notion of delegation in which all of the work items corresponding to a specific task definition can be redirected to another resource.

**3.5 Auto-start Patterns**

Auto-start patterns relate to situations where the execution of work items is triggered by specific events in the lifecycle of the work item or the related process definition. Such events may include the creation or allocation of a work item, completion of another instance of the same work item or a work item that immediately precedes the one in question. The state transitions associated with these patterns are illustrated in Figure 4.

In this section we describe two of the four patterns: piled execution (pattern 38) and chained execution (pattern 39). These correspond to transitions

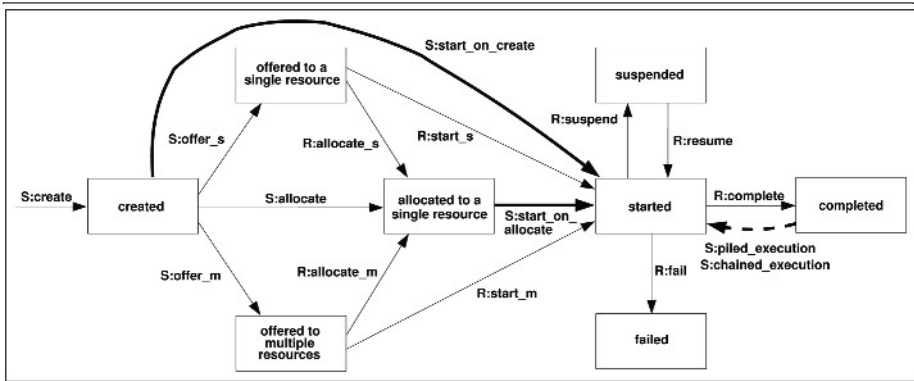


Fig. 4. Auto-start Patterns

`S:piled_execution` and `S:chained_execution` in Figure 4. Note that the corresponding arc is dashed because the `completed` state of one work item is connected to the `started` state of the next work item.

### Pattern 38: R-PE (Piled Execution)

**Description** The ability of the workflow system to initiate the next instance of a workflow task (perhaps in a different case) once the previous one has completed.

**Example** The next, possibly different, work item can commence immediately after the previous one has finished and it can be allocated to the same resource.

**Motivation** Piled execution provides a means of optimising task execution by pipelining instances of the same task and allocating them to the same resource. The resource undertakes work items sequentially and once a work item is completed, if another work item of the same type is present in the work queue, it immediately commences work on it – in effect it attempts to work on the next of the same types of work items. The aim with this approach to work distribution is to allocate similar work items to the same resource which aims to undertake them one after the other thus gaining from the benefit of exposure to the same type of task (e.g., reduced set-up time, increased familiarity with the task).

**Implementation** To implement this pattern requires like work items to be allocated to the same resource and the ability for the resource to undertake related work items on a sequential basis, immediately commencing the next one when the previous one is complete. This is a relatively sophisticated requirement and none of the workflow systems examined support it.

### Pattern 39: R-CE (Chained Execution)

**Description** The ability of the workflow system to automatically start the next work item in a case once the previous one has completed.

**Example** Immediately commence the next work item(s) in the workflow process when the preceding one has completed.

**Motivation** The rationale for this pattern is that case throughput is expedited when a resource is allocated sequential work items within a case and when a work item is completed, its successor is immediately initiated. This has the effect of keeping the resource constantly busy progressing work items in a given case.

**Implementation** In order to implement this pattern effectively, the majority (if not all) of the work items for a given case need to be allocated to the same resource and it must execute them in a strict sequential order. This approach to work distribution is best addressed by a case handling system and not surprisingly FLOWer offers direct support for it.

### 3.6 Additional Patterns

In [12] we identify additional patterns dealing with *visibility* (which work items are visible and who can see them) and *collaboration* (resources working on the same work item (e.g., patterns related to the formation of project teams)). In this paper, we will not elaborate on these patterns and we will also not include them in the product evaluation presented in the next section.

## 4 Evaluation of Existing Workflow Products

The resource patterns discussed in this paper have been collected for several reasons. First of all, the patterns serve as a way to describe workflow functionality in a tool-independent manner. Second, the patterns can be used to train workflow developers and consultants. Last but not least, the patterns can be used to select workflow products or other PAIS. In this paper, we focus on five products: Staffware Process Suite version 9 (TIBCO), WebSphere MQ Workflow 3.4 (IBM), FLOWer 3 (Pallas Athena), COSA 4.2 (TRANSFLOW) and iPlanet 6.0 (SUN). An assessment scale with three possible values is used for evaluating these products with “+” indicating direct support for the pattern, “+/-” indicating partial support and “-” indicating that the pattern is not supported. The specifics of the rating criteria used are described in [12]. Although “-” indicates that there is no (direct) support for the pattern, it does not imply that it is impossible to realise the pattern. Often it is possible to realise the functionality outside of the system, e.g., by calling an external program. However, such workarounds are not considered to be a feature of the product itself. See [2] for a similar distinction used when evaluating systems against control-flow patterns.

Lines 1 to 11 of Table 1 shows the support for creation patterns. Only the first two patterns are supported by all of the products examined.

Lines 12 to 20 of Table 1 report on push patterns. It is surprising to note that not all systems support simple allocation mechanisms such as round robin allocation. Lines 21 to 26 illustrate that pull patterns are supported by most systems. In particular, Staffware, FLOWer and COSA allow for greater initiative on the part of resources in selecting and commencing work items.

Detour patterns support is shown by lines 27 to 35 of Table 2. COSA provides resources with a significant degree of autonomy in managing their workload.



**Table 1.** Support for creation, push and pull patterns in workflow systems

Nr	Pattern	Staffware	WebSphere	FLOWer	CO5A	iPlanet
1	R-DA (Direct Allocation)	+	+	+	+	+
2	R-RBA (Role-based Allocation)	+	+	+	+	+
3	R-FBA (Deferred Allocation)	+	+	-	-	-
4	R-RA (Authorisation)	-	-	+	+	-
5	R-SOD (Separation of Duties)	-	+	+	+/-	+
6	R-CH (Case Handling)	-	-	+	-	-
7	R-RF (Retain Familiar)	-	+	+	+	+
8	R-CBA (Capability-based Allocation)	-	-	+	+	+
9	R-HBA (History-based Allocation)	-	-	-	+/-	+
10	R-OA (Organisational Allocation)	+/-	+	+/-	+	-
11	R-AE (Automatic Execution)	+	-	+	+	+
12	R-DBOS (Distribution by Offer – Single Resource)	-	-	-	+/-	+
13	R-DBOM (Distribution by Offer – Multiple Resources)	+	+	+	+	+
14	R-DBAS (Distribution by Allocation – Single Resource)	+	+	+	+	+
15	R-RMA (Random Allocation)	-	-	-	+	+/-
16	R-RRA (Round Robin Allocation)	-	-	-	+/-	+/-
17	R-SHQ (Shortest Queue)	-	-	-	+	+/-
18	R-ED (Early Distribution)	-	-	+	-	-
19	R-DE (Distribution on Enablement)	+	+	+	+	+
20	R-LD (Late Distribution)	-	-	-	-	-
21	R-RIA (Resource-Initiated Allocation)	-	-	+	+/-	-
22	R-RIEA (Resource-Initiated Execution – Allocated Work Item)	+	+	+	+	-
23	R-RIEO (Resource-Initiated Execution – Offered Work Item)	+	+	-	+	+
24	R-OBS (System-Determined Work List Management)	+	-	+	-	+
25	R-OBR (Resource-Determined Work List Management)	+	+	+	+	-
26	R-SA (Selection Autonomy)	+	+	+	+	+

Other workflow systems do not support the same degree of flexibility although it is worth noting that the case handling foundation of FLOWer lessens the need for such “detours” as work items within a case are generally handled by the same resource. Finally, lines 36 to 39 show the support for auto-start patterns. As was discussed in Section 3.5, it is remarkable that none of the systems support piled execution while only one of them supports chained execution.

**Table 2.** Support for detour and auto-start patterns

Nr	Pattern	Staffware	WebSphere	FLOWer	CO5A	iPlanet
27	R-D (Delegation)	+	+	-	+	-
28	R-E (Escalation)	+	+	-	+	+/-
29	R-SD (Deallocation)	-	-	-	+	+
30	R-PR (Stateful Reallocation)	+/-	+	-	+	-
31	R-UR (Stateless Reallocation)	-	-	-	-	-
32	R-S (Suspension)	+/-	+/-	-	+	-
33	R-SK (Skip)	-	-	+	+	-
34	R-REDO (Redo)	-	-	+	-	-
35	R-PRE (Pre-Do)	-	-	+	-	-
36	R-CC (Commencement on Creation)	-	-	-	+	-
37	R-CA (Commencement on Allocation)	-	+	-	-	-
38	R-PE (Piled Execution)	-	-	-	-	-
39	R-CE (Chained Execution)	-	-	+	-	-

## 5 Related Work

This paper is part of a bigger initiative (cf. [www.workflowpatterns.com](http://www.workflowpatterns.com)) to capture the functionality of PAIS in terms of patterns. It complements the control-flow [2] and data [11] patterns. This initiative provides a comprehensive pattern library for PAIS. There are a variety of other patterns libraries which target specific domains such as software engineering (<http://hillside.net/pattern>), interaction design (<http://www.welie.com>), and user interface and HCI design (<http://www.cs.ukc.ac.uk/people/staff/saf/patterns/gallery.html>).

Research on resource/organisational modelling in workflow systems has been relatively limited as discussed in [4, 8]. The RBAC (Role-Based Access Control) model [7] represents one approach for determining suitable users for a task. The salient features of RBAC are that permissions are associated with roles and users are made members of roles, thereby acquiring the associated permissions. RBAC models are useful but they have limitations, in particular they are primarily permission oriented, from a security point of view, and neglect other aspects of the organisation such as resource availability. Bussler and Jablonski [5] undertook pioneering work in the area, identifying many limitations of workflow systems in modelling policy and organisational issues. Several researchers [3, 9, 10] have developed so-called meta-models, i.e., object models describing the relationships between workflow concepts, including aspects of work allocation. However, these meta-models typically do not describe the dynamic aspects of work distribution.

## 6 Conclusion

In this paper we have presented a collection of resource patterns in the context of process-aware information systems. In addition, we have evaluated five workflow management systems using these patterns. We have only highlighted a selection of the available patterns and refer the reader to [12] for a complete description of all patterns and a detailed assessment of the five products. The main contribution of this work is that it is the first systematic analysis of the resource allocation/work distribution functionality desired in PAIS. The “patterns paradigm” was used to represent the different types of functionality. We do not claim completeness, but it is fair to say that the more than forty patterns identified provide a good coverage of the functionality provided by existing workflow management systems, as illustrated by the results presented in Section 4. Moreover, the patterns do not only apply to workflow technology but also to other process-aware systems, e.g., ERP systems such as SAP and PeopleSoft, which can be analysed and improved through their use.

## References

1. W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
3. W.M.P. van der Aalst and A. Kumar. Team-Enabled Workflow Management Systems. *Data and Knowledge Engineering*, 38(3):335–363, 2001.
4. W.M.P. van der Aalst, A. Kumar, and H.M.W. Verbeek. Organizational Modeling in UML and XML in the context of Workflow Systems. In H. Haddad and G. Papadopoulos, editors, *Proceedings of the 18th Annual ACM Symposium on Applied Computing (SAC 2003)*, pages 603–608. ACM Press, 2003.
5. C. Bussler and S. Jablonski. Policy Resolution for Workflow Management Systems. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, page 831. IEEE Computer Society, 1995.
6. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.0. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2002.
7. D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.
8. A. Kumar, W.M.P. van der Aalst, and H.M.W. Verbeek. Dynamic Work Distribution in Workflow Management Systems: How to Balance Quality and Performance? *Journal of Management Information Systems*, 18(3):157–193, 2002.
9. M. zur Muehlen. *Workflow-based Process Controlling: Foundation, Design and Application of workflow-driven Process Information Systems*. Logos, Berlin, 2004.
10. M. Rosemann and M. zur Muehlen. Evaluation of Workflow Management Systems - a Meta Model Approach. *Australian Journal of Information Systems*, 6(1):103–116, 1998.

11. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns. QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004. [http://www.citi.qut.edu.au/about/research\\_pubs/technical.jsp](http://www.citi.qut.edu.au/about/research_pubs/technical.jsp).
12. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Resource Patterns. BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004. <http://fp.tm.tue.nl/beta/>.
13. WFMC. Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface – XML Process Definition Language (XPDL) (WFMC-TC-1025). Technical report, Workflow Management Coalition, Lighthouse Point, Florida, USA, 2002. <http://www.wfmc.org/standards/docs.htm>.

# A Declarative Foundation of Process Models

Birger Andersson, Maria Bergholtz, Ananda Edirisuriya,  
Tharaka Ilayperuma, and Paul Johannesson

Department of Computer and System Sciences,  
Stockholm University/Royal Institute of Technology,  
Forum 100, SE-164 40 Kista, Sweden  
{ba, maria, si-ana, si-tsi, pajo}@dsv.su.se

**Abstract.** In this paper, a declarative foundation for process models is proposed. Three issues in process management and modeling are identified: business orientation, traceability, and flexibility. It is shown how these issues can be addressed by basing process models on business models, where a business model focuses on the transfer of value between agents. As a bridge between business models and process models, the notion of activity dependency model is introduced, which identifies, classifies, and relates activities needed for executing and coordinating value transfers.

## 1 Introduction

For information processing systems, it is possible to identify a number of functional aspects, [16]. The basic aspect is the services the system provides to its environment. By providing services, the system delivers value to its environment, which is the *raison d'être* of the system. The behaviour, or process, aspect has to do with the ordering and control flow of services and activities. The communication aspect concerns interactions with other entities, like people, hardware, and software. Finally, the meaning aspect addresses the interpretation of symbols used in services. The process aspect is attracting ever more attention, as witnessed by the advent of new workflow and process management systems, new process modeling languages, and dedicated journals and conferences. Despite their success, process technologies and in particular process modeling techniques face a number of shortcomings and challenges that need to be addressed. In this paper, we will address three of these:

- **Business Orientation.** Process models are typically expressed through low level concepts like control flow structures and message passing. Such concepts are not easily understood by business experts and users, who instead prefer to understand processes through business oriented notions like value exchanges and resource flows.
- **Traceability.** Constructing a process model includes taking a number of design decisions that affect the structure of the model. It should be possible to trace these design decisions back to explanations and motivations expressed in business terms.

- Flexibility. In most processes, the main structure is stable over time, while details may vary from case to case. Process models and systems should, therefore, allow for flexibility at design time as well as runtime.

One way to address these issues is to base process models on business models. By business model we mean a model which focuses on providing a high level view of the activities taking place by identifying agents, resources and the exchange of resources between the agents [12]. A process model, on the other hand, deals with operational and procedural aspects of business communication by focusing on the activities carried out by agents [1]. As the process model concentrates on technical details like *how* these activities will be carried out on the operational level, it becomes difficult for business users to understand it. Because of this it is hard for business experts and users to understand the connection between the high level view of the business model and the technical view of the process model.

The purpose of this paper is to propose a declarative foundation of process models based on business models. In order to bridge the gap between business models and process models, we introduce the notion of activity dependency model, which identifies, classifies, and relates activities needed for executing and coordinating value transfers. Having a declarative foundation will make it easier for designers to justify their design decisions at the technical level and trace them back to a business model. The paper builds upon and extends previous work, [1], by taking into account not only value transfer activities but also complimentary activities needed for their execution.

The paper is structured as follows. Section 2 gives an overview of related research. Section 3 gives an overview of business models. Section 4 introduces activity dependency models and shows how they are related to business models. Section 5 gives an overview of process models in a specific notation (BPMN [4]) and shows how they can be derived from activity dependency models. Finally, Section 6 concludes the paper and gives suggestions for further research.

## 2 Related Research

There exist several approaches to provide interfaces between technical concepts of software and systems design and business oriented requirements engineering. One of the most widely accepted approaches is the Unified Modeling Language, UML, with associated methodologies like RUP [13], where various types of design models represent static information in class diagrams as well as behavioral aspects modeled in interaction and activity diagrams. The UN/CEFACT Modeling Methodology (UMM) [15] uses UML as a base for specifying business processes involving information exchange in a technology-neutral and implementation-independent manner. A weak point in utilizing methodologies such as these, is still how to go from a business model to a process model in a systematic way. The DEMO methodology [5] was developed for the purpose of modeling essential business processes, abstracting completely from their realization. DEMO particularly stresses the distinction between information processes (generally modeled through low level message protocols) and the actual business processes, e.g. production acts where the agents fulfill the mission of the organization and coordination acts where agents enter into and comply with commitments. Dietz divides the collaboration between agents

into three distinct phases. The *Ordering phase*, in which an Agent requests some Resource from another Agent who, in turn, promises to fulfill the request. The *Execution phase*, in which the Agents perform Activities in order to fulfill their promises. The *Result phase*, in which an Agent declares a transfer of Resource control to be finished, followed by the acceptance or rejection by the other Agent. The ISO OPEN-EDI initiative [8] identifies five phases: Planning, Identification, Negotiation, Actualization and Post-Actualization. In this paper, we use only two phases: a *Negotiation phase* in which commitments are proposed and accepted, and an *Execution phase* in which transfers of Resources between Agents occur and are acknowledged.

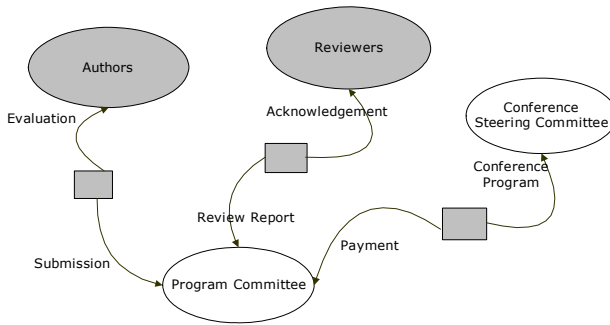
The approach proposed in this paper resembles DEMO since we also utilize the distinction between production processes and coordination processes. In our work this division among processes (or activities) is systematically derived from a business model through a set of activity dependencies, all expressed in business terms such as trust, commitments and transfer of value between agents [7] [11]. A similar approach may be found in the *i\** framework [17], which has been explored for modeling trust relationships among strategic actors. The activity dependency models of this paper may be used as a modeling means to reason about design decisions, just as the models are in *i\**. However, the specific aim of the activity dependency models here is to automate the process of going from business to process model.

In building process models, both an internal and external perspective on interactions between agents must be observed. In the Agent-Object-Relationship (AOR) approach [14], organizations and organizational behavior are modeled in two basic types of models: external and internal ones. An external AOR model adopts the perspective of an external observer who is observing agents and their interactions in the problem domain under consideration. In an internal AOR model, the internal (first-person) view of a particular agent to be modeled is taken. This notion is mirrored in our work, i.e. the activity dependency models introduced always take the view of one particular agent. Moreover, any interaction from one agent with respect to another agent has its counterpart in that other agent's activity model, i.e. one agent's internal activity is modeled as the other agent's external ditto. *PayInvoice* as an activity in one model has a reciprocal relationship with a *ReceivePayment* activity in the model of the corresponding agent. While AOR uses the internal business models as a point of departure to define so called interaction frames to describe the collaboration between agents, the work reported on in this paper suggests a way to automatically derive process models from business models. The proposed approach yields a process model that contains the main procedural logic. To get additional procedural details for sub-processes the notion of generic process patterns is used. The hypothesis is that most process models for organizational domains may be expressed as a combination of well documented design patterns [6] [9].

### 3 Business Models

The purpose of a business model is to describe and visualise the transfer of value between agents. A business model consists of three components: agents, value transfer offerings, and dualities. An agent is a person or (part of) an organisation that

is capable of controlling, acquiring, and providing resources. Examples of agents are consumers, companies, and government authorities. An agent may transfer resources to another agent in a value transfer, e.g. the delivery of a product or a payment. A value transfer offering represents the willingness of an agent to perform value transfers with other agents. In a business setting, it never happens that one agent simply transfers a resource to another agent - she always expects to get another resource back as compensation. As the saying goes, “one good turn deserves another”. To represent this reciprocity between value transfer offerings, we introduce the notion of duality. A duality associates two or more reciprocal value transfer offerings, e.g. the willingness to deliver a product and to pay for it.



**Fig. 1.** A Business Model

An example of a simple business model for the well-known scientific conference case is shown in Fig. 1. For space restrictions, we only consider the submission and acceptance part of the case. Authors (agent) can submit papers (value transfer offering) to the program committee (agent), who in return (duality) provides the authors with evaluations and decisions for acceptance (value transfer offering). In order to make the decisions, the program committee obtains review reports (value transfer offering) from reviewers (agents), who in return (duality) will get acknowledgements in the conference documentation (value transfer offering). The program committee will based on submitted papers and reviews deliver a conference program (value transfer offering) to the conference steering committee (agent), who in return (duality) provides financial reimbursement (value transfer offering). In Fig. 3.1, individual agents are represented by plain ovals, classes of agents by shadowed ovals, dualities by shadowed small rectangles, and value transfer offerings by arrows between dualities and agents (either individual agents or classes of agents). More precisely, a business model is defined as follows.

**Definition 3.1:** A business model is a directed graph with three types of nodes *IndAgt*, *ClassAgt*, *Dual* and directed edges called *VTO* from *Dual* to  $(IndAgt \cup ClassAgt)$ . *IndAgt*, *ClassAgt* and *VTO* are sets representing Individual Agents, Class Agents and Value Transfer Offering between these Agents, respectively. *Dual* is a set



of dualities where a duality  $d$  is a relation over  $(IndAgt \cup ClassAgt) \times VTO \times (IndAgt \cup ClassAgt)$  with the following property: if  $(x, y, z) \in d$  then  $\exists w \in VTO$  such that  $(z, w, x) \in d$  ■

## 4 Activity Dependency Models

### 4.1 Concepts and Notation for Activity Dependency Models

The purpose of an activity dependency model is to describe, on a high level, the activities needed for carrying out the value transfers specified in a business model. An activity dependency model provides more detail than a business model by identifying, classifying, and relating activities needed for executing and coordinating value transfers. On the other hand, an activity dependency model is less detailed than a process model, as it abstracts from ordering and control flow aspects. In this way, activity dependency models occupy a middle ground between business models and process models where they provide business-oriented information on activity coordination without going into procedural details. An activity dependency model is always constructed from a particular agent's perspective, called the base actor, i.e. the model takes the internal view as discussed in Section 2. This means that an activity dependency model focuses on one agent in a business model and the dualities involving this agent.

Structurally, an activity dependency model can be seen as a graph with four kinds of nodes, representing activities, and four kinds of directed edges, representing relationships between activities. The four kinds of activities are:

- *Value transfer activities.* A value transfer activity transfers resources from one agent to another and corresponds directly to the value transfer offerings in a business model.
- *Assignment activities.* An assignment activity relates a specific agent from a class of agents to the value transfer activities of one duality. An example is the assignment of a reviewer for reviewing a particular paper.
- *Production activities.* In a production activity, the base actor produces a resource required for a value transfer activity.
- *Coordination activities.* A coordination activity coordinates the value transfer activities within one duality as well as additional assignment and production activities.

The four kinds of relationships between activities are:

- *Duality dependencies.* A duality dependency from a coordination activity to a value transfer activity expresses that the latter is included in the duality of the former; recall that each coordination activity corresponds to one duality.
- *Flow dependencies.* A flow dependency, [10], from one activity to another expresses that the resource obtained by the first activity is needed as input to the second activity. An example is a retailer who has to obtain a product from an importer before delivering it to a customer.

- *Trust dependencies.* A trust dependency [1], between two value transfer activities within the same duality expresses that the first activity has to be carried out before the second one as a consequence of low trust between the involved agents. Informally, a trust dependency states that one agent wants to see the other agent do her work before doing his own work. An example could be a car dealer requiring a down payment from a customer before delivering a car.
- *Trigger dependencies.* A trigger dependency from a coordination activity to an assignment or production activity expresses that the latter is to be initiated and managed by the coordination activity.

The components of an activity dependency model have a clear business motivation, i.e. they can be explained and motivated in business terms. This makes the activity dependency model a useful instrument for eliciting and communicating business knowledge. At the same time, an activity dependency model provides an adequate basis for constructing more detailed process models, as will be discussed in the next section.

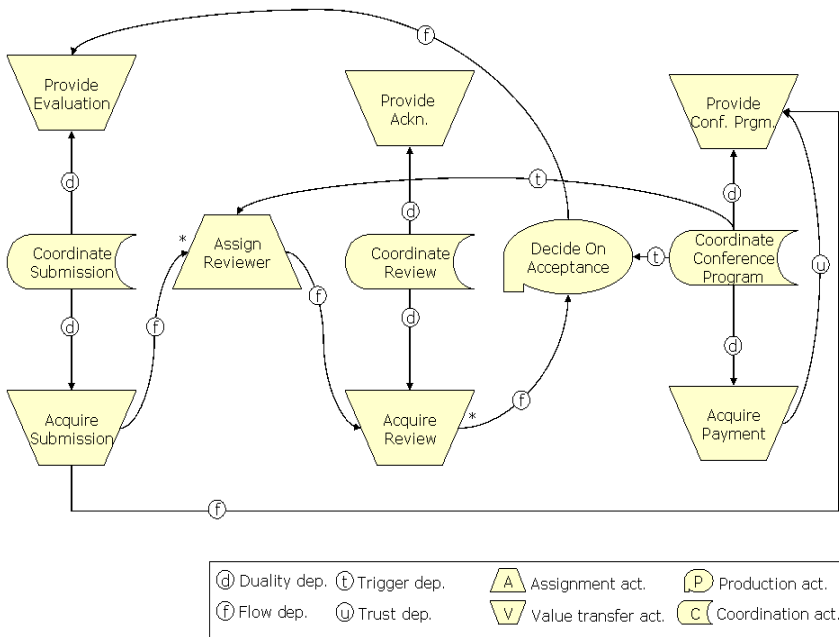


Fig. 2. An Activity Dependency Model

An example of an activity dependency model is shown in Fig. 2, which is based on the business model from the previous section. The base actor is the program committee, and the diagram shows three columns of coordination and value transfer activities corresponding to the dualities of this actor. There is one assignment activity *AssignReviewer* (for assigning a reviewer to a paper), one production activity *DecideOnAcceptance* (for deciding whether to select a paper), and a number of flow,

trust, and trigger dependencies. For example, there is a flow dependency from *AssignReviewer* to *AcquireReview* meaning that an assignment of a reviewer to a paper must exist before a review of that paper can be obtained. Furthermore, cardinalities (as in UML) have been attached to some of the dependencies. For example, the star on the flow dependency from *AcquireSubmission* to *AssignReviewer* means that for each submitted paper, several assignments of reviewers may exist.

**Definition 4.1:** An activity dependency model is a directed graph with four types of nodes *VTA*, *Coord*, *Ass*, *Prod* and four types of directed edges *Dual*, *Flow*, *Trust*, and *Trigger*; where *VTA*, *Coord*, *Ass*, and *Prod* are sets representing value transfer activities, coordination activities, assignment activities, and production activities, respectively and *Dual*, *Flow*, *Trust* and *Trigger* are relations such that:

- *Dual* is an injective relation over  $Coord \times VTA$ .
- *Flow* is a relation over  $(VTA \cup Ass \cup Prod) \times (VTA \cup Ass \cup Prod) \times \{1, *\} \times \{1, *\}$  such that, if  $(x, y, z, w) \in Flow$  then  $(x, y) \in \{(VTA \cup Ass \cup Prod) \times (VTA \cup Ass \cup Prod) \setminus (Ass \times (Ass \cup Prod))\}$  and  $(z, w) \in (\{1, *\} \times \{1, *\})$
- *Trust* is a relation over  $VTA \times VTA$  within the same duality
- *Trigger* is a relation over  $Coord \times (Ass \cup Prod)$  ■

## 4.2 From Business Model to Activity Dependency Model

An activity dependency model can partially be derived from the business model it is based on. Each duality in the business model gives rise to one coordination activity, and each value transfer in the business model gives rise to one value transfer activity. The value transfer activities within one duality are related to the corresponding coordination activity by duality dependencies. Furthermore, for dualities where the base actor may choose between several agents in a class of agents (for example choosing a reviewer for a paper), an assignment activity is added. For each duality, one production activity may also be added when the base actor has to produce some resource needed for a value transfer activity. These components of the activity dependency model are directly derived from the business model, but the activity dependency model will also contain a number of additional components not derivable from the business model, namely the flow, trust, and trigger dependencies.

**Definition 4.2:** Let  $BM = \langle IndAgt, ClassAgt, Dual, VTO \rangle$  be a business model. Let  $AM = \langle VTA, Coord, Ass, Prod, Dual, Flow, Trust, Trigger \rangle$  be an activity dependency model. *AM conforms to BM* if

- for each duality in *Dual* there is one coordination activity in *Coord*
- for each value transfer in *VTO* there is one value transfer activity in *VTA*
- for each element  $d = \langle x, y, z \rangle$  in a duality in *BM-Dual* there is a pair  $\langle c, v \rangle$  in *AM-Dual*, where  $c$  is the coordination activity corresponding to  $d$  and  $v$  is the value transfer activity corresponding to  $y$ , and a pair  $\langle c, w \rangle$  where  $c$  is the same coordination activity and  $w$  is a value transfer activity corresponding to  $z$ .

- for each  $d$  in *Dual*, there is optionally one production activity in *Prod*
- for each  $d$  in *Dual* related to an element in *ClassAgt*, there is one assignment activity in *As* ■

## 5 Process Models

### 5.1 Concepts and Notation for Process Models

The notation we will use for process models is BPMN [4], a standard developed by the Business Process Management Initiative (BPMI) [3]. The goal of BPMN is to be an easily comprehensible notation for a wide spectrum of stakeholders ranging from business domain experts to technical developers. A feature of BPMN is that BPMN specifications can be readily mapped to executable XML languages for process specifications such as BPEL4WS [2].

In this paper, we will use only a selected set of core elements from BPMN. These elements are Activities, Events, Gateways, Sequence Flows, Message Flows, Pools and Lanes. Activity is a generic term for work that an Agent can perform. In a BPMN diagram, an Activity is represented by a rounded rectangle. An Activity can be atomic or compound. A Compound Activity is composed of other Activities and will be marked by a '+' sign inside the rounded rectangle. An Activity may also be repeated, which is graphically shown by a circular arrow inside the rounded rectangle. Events, represented as circles, are something that "happens" during the course of a business process. There exist three types of Events: Start, End and Intermediate Events. Activities and Events are connected by Sequence Flows, shown as arrows, indicating the order in which Activities will be performed in a business process. Gateways are used to control the Sequence Flows by determining branching, forking, merging, and joining of paths. In this paper we will restrict our attention to XOR and AND branching, graphically shown as a diamond with an 'X' or '+', respectively. Pools are graphical constructs, in the form of oblong rectangles enclosing other BPMN elements, for separating different sets of activities from each other. Message flows, shown as dotted arrows, are used for communication between Activities in different Pools. An example of a BPMN process diagram is given in Fig. 3.

### 5.2 From Activity Dependency Model to Process Model

Moving from an activity dependency model to a process model is essentially about specifying the detailed control flow between activities. The starting point is to let each coordination activity in the activity dependency model become a process defined within one pool. This means that each pool models the exchange of resources between the base actor and one other actor as well as the assignment and production activities needed for this exchange. The entire process model will consist of a number of such processes within pools that communicate with each other.

A single pool essentially describes a binary collaboration between two partners. Such a collaboration may consist of several phases as discussed in Section 2, including planning, identification, negotiation, actualisation, and post-actualisation. In

this paper, we consider only two phases: one Negotiation phase in which commitments for resource exchanges are proposed and accepted, and one Execution phase in which resource transfers occur and are acknowledged. Typically, a process will contain both a Negotiation phase and an Execution phase, but in some cases only the Execution phase is included. If the Negotiation phase is included, one sub-process for this phase is added to the pool. Thereafter, for each value transfer activity associated to the coordination activity, one sub-process is added. Furthermore, if there are any trigger dependencies from the coordination activity to assignment or production activities, one sub-process is added for each related assignment or production activity. There are also additional sub-processes for acquiring relevant resources, as specified through flow dependencies. Some of the sub-processes within a pool will be repeating. This occurs when there is a multi-valued flow dependency to the activity corresponding to the sub-process. The sub-processes are related by adding sequence flows between pairs of sub-processes if there is a flow or trust dependency between the corresponding activities. Finally, the negotiation sub-process is related to the other sub-processes by an AND-gateway.

As each coordination activity gives rise to its own pool, we will end up with a number of processes in pools that have to be connected to each other via message flows. Two pools need to be connected if one of the pools requires a resource provided by the other. There will be one message flow from the resource providing pool to the resource requesting pool, which informs about the delivery of the resource. Furthermore, there will be one message flow from the resource requesting pool to the resource providing if the latter contains a negotiation phase; this message flow is the request for the resource.

An example of a process based on the coordination activity *Coordinate ConferenceProgram* in Fig. 2 is shown in Fig. 3. It is assumed that this process does not contain a Negotiation phase but only an Execution phase (for reasons of completeness we also include the processes based on the other two coordination activities from Fig. 2 *Coordinate Review* and *Coordinate Submission* in Fig. 3, where *Coordinate Review* is assumed to contain a Negotiation phase). The two value transfer activities associated to *CoordinateConferenceProgram* give rise to two sub-processes *ProvideConferenceProgramme* and *AcquirePayment*. These and other sub-processes may have a more or less complex internal structure, and we will return to this issue in Section 5.3. Two more sub-processes are added, *AssignReviewer* and *DecideOnAcceptance*, based on the trigger dependencies in the activity dependency model. Furthermore, sub-processes that acquire relevant resources need to be added, in this case *ReceiveSubmission* and *GetReview*. *ReceiveSubmission* is added as there is a flow dependency from *AcquireSubmission* to *ProvideConferenceProgram* in the activity dependency model. *GetReview* is added as there is a flow dependency from *AcquireReview* to *DecideOnAcceptance*. Finally, a sub-process *DeliverEvaluation* is added due to the flow dependency from *DecideOnAcceptance* to *ProvideEvaluation*. The sequence flows in the diagram are all derived from flow dependencies, except the one from *AcquirePayment* to *ProvideConferenceProgram*, which is based on a trust dependency. Fig. 3 shows the entire process model for the conference case with message flows included.

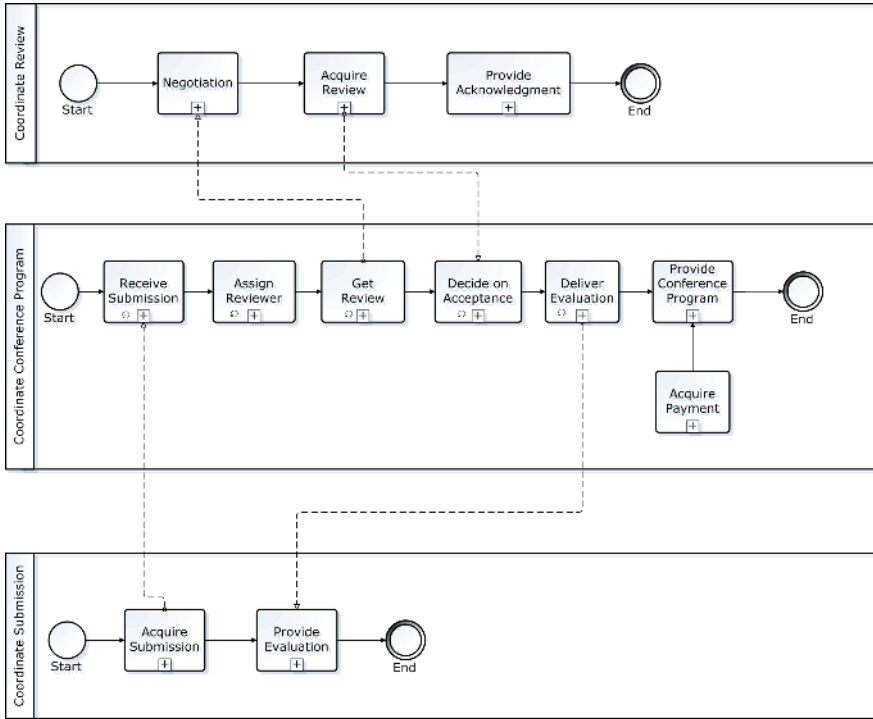


Fig. 3. A Process Model in BPMN

**Definition 5.1:** Let  $P$  be a pool containing an activity  $A$ . We will denote that activity by  $P.A$ . ■

**Definition 5.2:** Let  $AM = \langle VTA, Coord, Ass, Prod, Dual, Flow, Trust, Trigger \rangle$  be an activity dependency model. A coordination activity  $C$  in  $Coord$  will be related through duality dependencies to one or several value transfer activities in which the base actor provides resources to another actor. These value transfer activities will be denoted  $Out_1, \dots, Out_n$ . Analogously, the value transfer activities where the base actor acquires resources from another actor will be denoted  $In_1, \dots, In_n$ . ■

**Definition 5.3:** Let  $AM = \langle VTA, Coord, Ass, Prod, Dual, Flow, Trust, Trigger \rangle$  be an activity dependency model. Let  $PM$  be a BPMN diagram.  $PM$  conforms to  $AM$  if for each coordination activity  $C$  in  $Coord$  there is one pool  $P_C$  in  $PM$  fulfilling the following conditions:

- $P_C$  contains optionally one sub-process entitled *Negotiation*
- For each value transfer activity  $In_i$ , there is one sub-process  $In_i$
- For each value transfer activity  $Out_i$ , there is one sub-process  $Out_i$
- For each flow dependency from an activity  $D.In_i$  (where  $D$  is a coordination activity  $\neq C$ ) to a value transfer activity  $Out_j$ , there is one subprocess  $get(D, In_i)$ .

The sub-process is repeating if the flow dependency is not injective. There is one message flow from  $P_D.In_i$  to  $get(D.In_i)$ . If  $P_D$  contains a sub-process *Negotiation*, then there is also one message flow from  $get(D.In_i)$  to  $P_D.Negotiation$ .

- For each trigger dependency from  $C$  to an assignment or production activity  $Act$ , there is one sub-process  $do(Act)$ . The sub-process is repeating if there is a flow dependency from a value transfer activity  $D.In_i$  to  $Act$  and  $get(D.In_i)$  is included in  $P_C$  and repeating, or if there is a flow dependency from  $Act$  to  $C.Out_i$  that is multi-valued.
- For each flow dependency from an activity  $Act$  to a production activity  $Prod$  such that the corresponding sub-process  $do(Prod)$  is included in  $P_C$ , there is one sub-process  $do(Act)$ . The sub-process is repeating if  $do(Prod)$  is repeating, or if the flow dependency is not injective.
- If there is a production activity  $Prod$  with a flow dependency to a value transfer activity  $D.Out_i$  and the sub-process  $do(Prod)$  corresponding to  $Prod$  is included in  $P_C$ , then there is a message flow from  $do(Prod)$  to  $D.Out_i$ .

There is a sequence flow between two sub-processes if there is a flow dependency or trust dependency between the corresponding activities. When all sub-processes have been linked in this way, there will be a number of leftmost sub-processes,  $L_1, \dots, L_m$ , i.e. sub-processes with no incoming sequence flow. There will be a gateway  $G$  in  $P_C$  with a sequence flow from *Negotiation* to  $G$ . There will be sequence flows from  $G$  to each of  $L_1, \dots, L_m$ .

Finally, if a process model  $PM$  conforms to an activity dependency model  $AM$  and  $PM'$  is derived from  $PM$  as specified below, then  $PM'$  conforms to  $AM$ .  $PM$  should contain two repeating sub-processes  $S$  and  $T$  joined by a sequence flow.  $PM'$  is derived from  $PM$  by replacing the repeating sub-processes and their joining sequence flow by one repeating sub-process containing  $S$  and  $T$  joined by a sequence flow. ■

The reason for the last paragraph of the definition is to allow for more flexibility in the process design. Without it, there would be an unwanted restriction, namely how repeating activities are treated. It would be assumed that if there are two subsequent repeating activities then there must be a repetition over the first activity followed by a repetition over the second. For example, in Fig. 3 all submissions have to be received before any assignment is made. However, this assumption is too restrictive in many

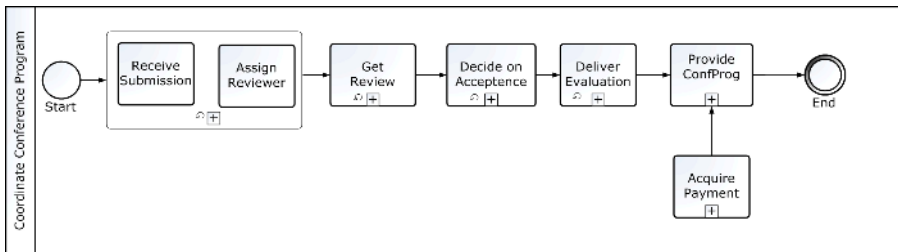


Fig. 4. An alternative Process Model in BPMN

cases, as an alternative is to have a single repetition over a sequence of the two activities. For example, in Fig. 4 a received submission may be directly followed by an assignment.

Due to space limitations we have only considered the success path, or “happy path”, of a process, i.e. we have not included exceptions, failed negotiations, etc. Extensions for these possibilities are obviously needed, but they are standard and do not affect the basic relationships between activity dependency models and process models.

### 5.3 Process Patterns

A process model derived from an activity dependency model, as described above, contains only the main procedural logic. Additional procedural details for the sub-processes of the process model are needed. One approach is to treat sub-models as the main model, i.e. detailed views of the main business model is provided and sub-activities and sub-processes derived according to the transformation described in the previous section. We believe, however, that a more flexible way to provide the procedural details is to use the notion of process patterns. The idea is that each sub-process is based upon a generic process pattern.

#### Generic Process Patterns

UN/CEFACT has defined a number of business transaction patterns as part of UMM with the intention of providing an established semantics of frequently occurring business interactions. Individual sub processes such as, for example, negotiations and fulfillments of earlier commitments to perform value transfers are easily expressed or assembled on an arbitrary level of granularity through the usage of generic transaction patterns. Below we list a number of patterns from [15] and [1].

#### Negotiation Phase Patterns

The *Contract proposal* [11] transaction pattern models the non-legally binding negotiation phase in a contract formation, whereas the *Commercial (Offer-Accept)* [15] transaction pattern expresses the formal creation phase of a contract. These patterns may be assembled into more complex patterns of collaborations between partners. An example of the latter is the UMM “simple negotiation pattern”, which combines variants of proposals and offers for contracts of transfer of value resources. The assembling of patterns is based on a layered approach where each base pattern constitute a sub-process in the assembled pattern, and where the “happy path” sequence flow of each pattern serves as connector between the pattern.

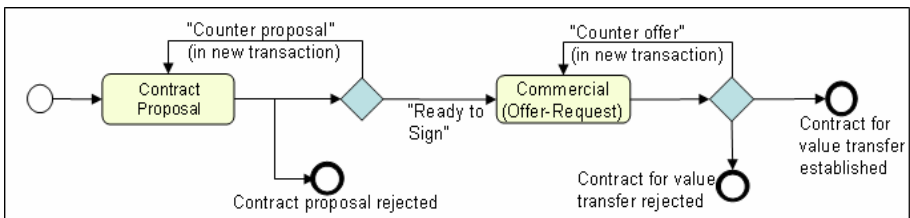
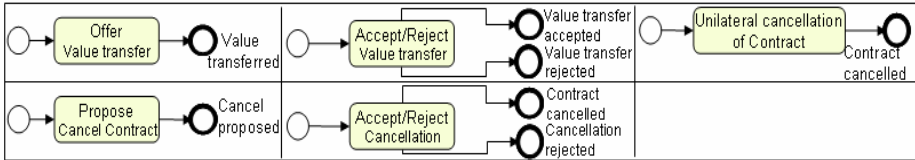


Fig. 5. Simple Negotiation pattern [15]

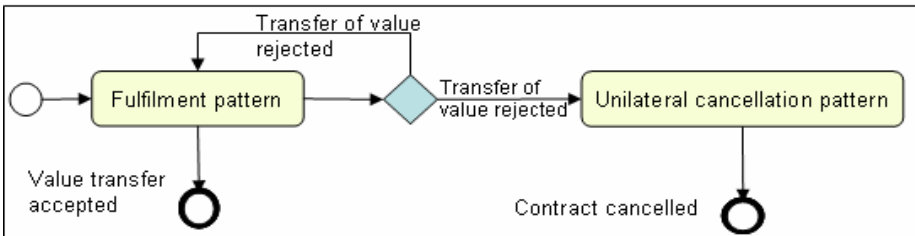


**Execution Phase Patterns**

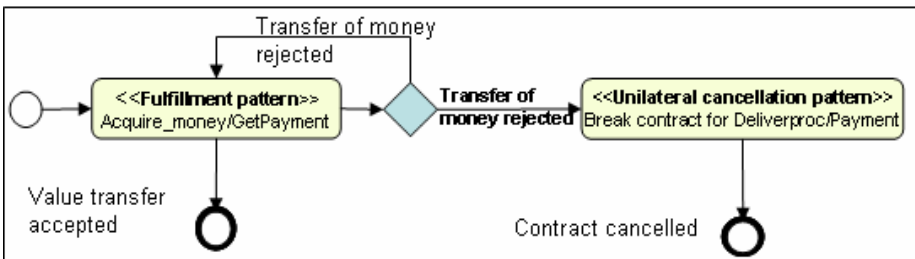
The fulfillment pattern specifies the completion of a Value transfer between two partners. The Bi- and Unilateral Cancellation transaction patterns refer to the unilateral or bilateral cancellation of a contract or to commitment(s) within a contract for value transfers.



**Fig. 6.** Fulfillment, Fulfillment Accept/Reject, bilateral and unilateral cancellation transaction patterns [1]



**Fig. 7.** Fulfillment collaboration pattern [1]



**Fig. 8.** Instantiation of Fulfillment collaboration pattern

The fulfillment collaboration pattern specifies relevant transaction patterns (Fig. 7) and the rules for transitioning among these within the completion of a value transfer. The pattern is assembled from the Fulfillment and Unilateral Cancellation transaction patterns defined in the previous section.

Finally, as an example of an instantiation of a generic collaboration pattern, consider the case of sub-process “Get-payment” from the Conference case, where the procedural details may be defined in a flexible way through combinations of generic transactions patterns. In fig 8 the “Get-payment” sub-process is modeled through the

Fulfillment collaboration pattern, which specifies relevant transaction patterns and the rules (variants of the rules for when and how to break a contract of value transfer amounts to choosing the right combination of patterns) for transitioning among these within the completion of a value transfer.

## 6 Conclusions and Further Work

In this paper, we have proposed a declarative foundation of process models based on business models. A key element of the approach is the activity dependency model, which works as a bridge between a business and a process model. We believe that this approach effectively addresses the issues introduced in Section 1:

- **Business Orientation.** Instead of going directly into procedural details, an activity dependency model allows business experts and users to describe the underlying business reasons that govern the flow of processes. In particular, relations between activities can be specified in terms of notions like resource flow, trust, coordination, and reciprocity.
- **Traceability.** A process model is based on an activity dependency model, which in turn is based on a business model. This means that components in a process model can be explained by and tracked back to business oriented notions and motivations.
- **Flexibility.** The transformations from business model to activity dependency model to process model give the main structure of a process. However, the approach allows for flexibility by letting sub-processes be based on patterns. This means that the lower-level details of a process model can be tailored to the situation at hand by selecting appropriate patterns from a repository.

We have introduced the notion of activity dependencies for capturing relationships between the activities within a process. Four kinds of dependencies have been identified, flow, trust, trigger, and duality dependencies. These can be stated declaratively, have a clear business motivation, and are used for the construction of the process model. A topic for further work is to investigate whether additional kinds of activity dependencies are required. Another topic for further research is to study how to include more phases in the process models, in addition to the negotiation and execution phases.

## References

1. Bergholtz M., Jayaweera P., Johannesson P., Wohed P., "A pattern and dependency based approach to the design of process models", in Proc. of the 23rd International Conference on Conceptual Modeling(ER2004), Shanghai, China
2. Business Process Execution Language for Web Services, OASIS WS-BPEL Technical Committee, Valid on 20041115, <http://www.ebpmi.org/bpel4ws.html>
3. Business Process Management Initiative (BPMI), Valid on 20041115, <http://www.bpmi.org/>

4. Business Process Modelling Notation (BPMN), Valid on 20041115, <http://www.bpmn.org/>
5. Dietz J.L.G, Deriving “Use Cases from Business Process Models”, ER2003, LNCS2813, pp.131-143, L-Y Song et al. (Eds.), Springer-Verlag Berlin Heidelberg 2003
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns*, Addison-Wesley, 1995
7. Gordijn J., Akkermans J. M. and Vliet J. C., “Business Modelling, is not Process Modelling”, Proc. of the 1th International Workshop on Conceptual Modeling Approaches for e-Business (eCOMO’2000), held in conjunction with the 19th International Conference on Conceptual Modeling (ER’2000), Salt Lake City, Utah, USA
8. *Open-EDI phases with REA*, UN-Centre for Trade Facilitation and Electronic Business, Valid on 20040419, [http://www.unece.org/cefact/docum/download/02bp\\_rea.doc](http://www.unece.org/cefact/docum/download/02bp_rea.doc)
9. Larman, C., “Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development”, 3rd Edition, 2004, ISBN: 0131489062
10. Malone et al.: “Towards a handbook of organizational processes”, MIT eBusiness Process Handbook, Valid on 20040419, <http://ccs.mit.edu/21c/mgtsci/index.htm>
11. Osterwalder, A, Parent, C., and Pigneur, Y., “Setting up an ontology of business model”, CAISE/EMOI’2004 (INTEROP workshop), 2004
12. McCarthy W. E., “REA Enterprise Ontology”, Valid on 20040419, <http://www.msu.edu/user/mccarth4/rea-ontology/>
13. Rational Unified Process (RUP) Valid on 20041115, <http://www-306.ibm.com/software/awdtools/rup/>
14. Wagner, G.,. “The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior”. *Information Systems* 28:5 (2003)
15. UN/CEFACT Modeling Methodology (UMM-N090 Revision 10), Valid on 20040419, [http://webster.disa.org/cefact-groups/tmg/doc\\_bpwg.html](http://webster.disa.org/cefact-groups/tmg/doc_bpwg.html)
16. Weiringa, R., Blanken, H., Fokkinga, M. , Grefen, P., Aligning Application Architecture to Business Context, Caise 2003, LNCS 2681 pp. 209- 225
17. Yu, E., Liu, L., “Modelling Trust in the *i\** Strategic Actors Framework” Proceedings of the 3rd Workshop on Deception, Fraud and Trust in Agent Societies. Barcelona, Catalonia, Spain (at Agents2000), June 3-4, 2000

# Synchronizing Copies of External Data in Workflow Management Systems

Johann Eder and Marek Lehmann

Alps-Adria University Klagenfurt,  
Department of Informatics-Systems  
{eder, marek}@isys.uni-klu.ac.at

**Abstract.** Workflow management systems integrate applications and data resources in business processes. Frequently they have to keep local copies of data in the so called workflow repository. We introduce and define synchronization policies for these copies with their external data sources ranging from the provision of fully synchronized replications of external data sources to unsynchronized storage of read results. We analyze in detail the effects of combining various pull and push policies and show in an example how these policies can be used in different situations.

## 1 Introduction

Workflow management systems (WfMSs) [7, 15] are instrumental in automating business processes, enacting activities due to business logic represented in workflow definitions and documenting the execution history. Workflow systems also frequently integrate separate information systems, using and manipulating data from various sources. Nevertheless, research on access to and integration of data by workflow systems is scarce compared with the intensive research on the control flow aspects [3].

A WfMS coordinates the execution of different Wf activities which can access different data sources. To determine the state transitions of a workflow (e.g. transition conditions) the WfMS uses  $\dots$  [15], which may be accessed both by the WfMS and the applications. Usually, the WfMS needs direct access to the workflow relevant data and, therefore, stores these data in an internal workflow repository. This raises the issue of synchronizing the replicas in the workflow repository with their originals, which typically may be altered by other applications unnoticed by the workflow system. Being unaware of possible synchronization problems (e.g. decisions based on stale copies of data, lost updates, unawareness that data needed for past decisions has changed, timely propagation of changes to copies, etc.) leads to potential application errors.

Synchronization of copies of external data in workflow repositories is, however, not trivial. Workflows might be long-running. Some copies need be refreshed, others not. Changes might be local or might need externalization, etc. Mere replication management is not sufficient, since it is frequently not adequate and more sophisticated synchronization policies are required to allow workflow

designer to achieve the required behaviour. We would like to provide synchronization based on policies, such that the designer can chose a synchronization policy and thus select the exact semantics and properties of data access operations.

Currently, access to external data is typically provided by invoking automated activities or other means which are not a part of the WfMS (e.g. Automatic Steps and Scripts in Staffware [13] or Java code attached to the activities in @enterprise [8]). The problem is that the programmers have to hardcode the mechanisms for accessing external data in automated activities. A workflow definition can contain a hundred or more activities, use data from dozens of external data sources like legacy systems, web services etc. Therefore, additional activities responsible only for keeping a copy of some data in the workflow repository increases the complexity of the workflow definition up to date. Such workflow definitions become difficult to maintain, in particular, when it is necessary to add new data sources or replace exiting ones. It may not always be clear whether an activity overwrites a local copy with the data from an external source or pushes local changes from the workflow repository to the environment. If such activities are tailored to one workflow definition, then it is difficult to reuse them in an other definition. These are activities which serve a purely technical purpose and do not represent any step in the business procedure. These activities, furthermore, have subtle relationships and interdependencies with other activities increasing the complexity of maintenance and evolution.

We propose an abstraction layer for WfMS which allows transparent access to any data source [6]. This is achieved by creating reusable and interchangeable wrappers around external data sources, which present to the WfMS the content of underlying data sources, manage the access to it, and provide thus also the basic synchronization operations. The functionality of the external data source is abstracted in these plug-ins. We introduced the workflow language WDL-X (an extension of WDL [5]) which makes use of these plug-ins and provides a coherent seamless data view of workflow data and external data.

In this paper we extend WDL-X with a transparent and manageable mechanism for maintaining copies of external data in a workflow repository.

The remainder of this paper is organized as follows: Section 2 describes new mechanisms for copying external data into the workflow repository in our workflow definition language WDL-X. In Section 3 we discuss in detail different maintenance policies for such copies. An example application of these policies is presented in Section 4. We discuss related work in Section 5 and finally draw some conclusions in Section 6.

## 2 Copying External Data into a Workflow Repository with the WDL-X

### 2.1 WfMS Architecture Including Data Access Plug-Ins

The work we present here is a continuation of our work in workflow systems. Our workflow system, *Staffware* [5] used a form-flow metaphor to provide access to

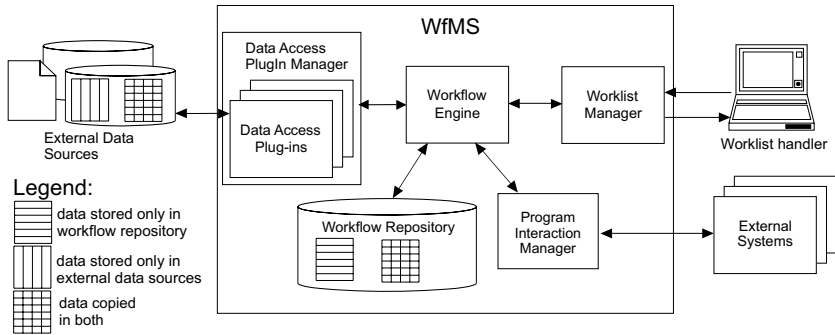


Fig. 1. WfMS architecture

workflow specific data. In [6] we proposed a uniform treatment of all kinds of business data in a workflow definition. We used XML as data access language in our workflow definition language WDL-X [6], which replaced the earlier WDL [5]. The transparency of data location and logical and physical data independence of workflow systems is achieved in WDL-X by using specialized wrappers around external data sources called data access plug-ins. In this paper we describe an extension of the WDL-X and the WfMS functionality, which allows to manage copies of external data in the workflow repository.

The data access plug-ins introduced in WDL-X provide an abstraction layer between the WfMS and the actual storage and format of the data. WDL-X uses XML as unifying data format which allows to integrate data from external sources, data exchange between workflow systems or web services, and internal data handling. A data access plug-in is a wrapper presenting to the WfMS the content of external data sources as documents of a predefined XML Schema type. Data sources might be legacy systems, relational or object relational DBMSs, files stored in a file system or native XML databases etc., both in- and outside of an enterprise. A data access plug-in exposes to the WfMS a simple interface for the creation, selection, update and deletion of an XML document in a collection of many documents of the same XML Schema type and the evaluation of the XPath expressions on a selected document. The task of the plug-in is to translate these operations on XML documents to the underlying data sources. A workflow designer can specify in a workflow definition, which document should be accessed by which data access plug-in. The WfMS calls the associated data access plug-in each time an operation on a specified XML document is performed. This provides the transparency of actual data location and allows to use external data to control the flow of a workflow. Data access plug-ins are collected in a library and might be used in several workflows.

The overall WfMS architecture is presented in Fig. 1. The workflow engine is responsible for execution of instances of workflows based on process definitions. Both process definitions and instances are stored in the workflow repository. The worklist manager is responsible for the worklists of human actors and for the interaction with the client software (worklist handlers). The program interaction

manager calls programs implementing automated activities. The workflow engine can access data stored either in the workflow repository or in external data sources. The engine uses data access plug-ins to transparently access external data.

## 2.2 Workflow Repository and External Data

Sometimes it is necessary that a WfMS works on a copy of external data stored in the workflow repository. The synchronization needs of these copies are quite different, depending on the application. In some cases it may be required that the WfMS operates on a stale copy, e.g. during a processing of some application the marital status of an applicant at the starting time of the process is important, even if the status has changed in the meantime. In other cases it may be required to restrict the possibility to externalize modifications made locally to the copies in the workflow repository, e.g. these modifications cannot be propagated to the original data until they are proven correct. Therefore, we argue that a WfMS needs a transparent and manageable mechanism for making and synchronizing copies of external data. We propose to use the functionality of data access plug-ins to copy external data into the workflow repository combined with a set of different policies for synchronizing these copies.

External data and their copies in the workflow repository are independent. External data may be read and updated by external systems and their copies in the workflow repository may be read and updated by the workflow instances. A workflow designer has to define in a WDL-X script a desired policy for synchronizing a copy with its original:

- Refresh policy for a copy in the workflow repository:
  - **refresh** – the WfMS automatically refreshes the copy with external data before each read access to the copy made within a process instance,
  - **refreshOnDemand** – the workflow designer has to use explicitly the WDL-X command `forceRefresh` to refresh the copy,
  - **doNotRefresh** – keeps the copy isolated from the changes made in the original data.
- Push policy for changes made locally in the workflow repository:
  - **immediatePush** – the WfMS automatically pushes to the environment each change made to the copy in the workflow repository,
  - **pushOnDemand** – the workflow designer has to use explicitly the WDL-X command `forcePush` to push the copy,
  - **doNotPush** – keeps all the changes locally in the workflow repository.

To refresh and to push a copy the WfMS uses a data access plug-in associated with a document copied from an external data source. The refresh mechanism depends on the capabilities of a data access plug-in and an underlying data source. A `FileAccess` plug-in is not capable of monitoring original data for changes and, therefore, the WfMS has to use the plug-in to reload the original document each time it refreshes a copy. An `DatabaseAccess` plug-in is able to monitor original data for changes (e.g. using triggers in an underlying database) and reload the original data only if these data were changed.

### 2.3 Workflow Definitions with Data Access Plug-Ins

In WDL-X the process variables are bound to XML documents of predefined XML Schema types. A variable can be bound either to an existing document by the WDL-X command `openDocument` or to a new document created during a process execution. If no document is bound to a variable, then the variable is not initialized. The process variables are local to a process instance. Activities can receive parameters in one of the following modes: - as an input parameter, - as an output parameter and - as an in- and output parameter. Conditions may be tested on a process variable, e.g. to evaluate the control flow of a workflow.

Based on this description an example variable declaration in WDL-X looks as follows:

```
documents customerData : customerType accessedBy customerDbPlugIn
                        doNotRefresh doNotPush;
```

The semantics of this example is: In a workflow definition a variable named `customerData` of the XML Schema type `customerType` is declared. The variable should be bound after the initialization to an XML document created as a copy of a document accessed by a data access plug-in named `customerDbPlugIn`. The copy stored in the workflow repository should not be refreshed and the changes made to the copy should not be pushed back to the original document.

## 3 Policies for Synchronizing Copies of External Data in WfMSs

Documents bound to process variables may be stored in the external data sources, be stored in the workflow repository, or the workflow repository contains a copy of external data (see Fig. 1). In this paper we focus only on the latter case.

### 3.1 Basic Operations on Documents

We refer to the set of documents stored in the external systems as  $X$ , and  $\tilde{X}$  denotes the set of documents stored in the workflow repository:

$$X = \{x : x \text{ is a document stored in an external data source}\} \quad (1)$$

$$\tilde{X} = \{\tilde{x} : \tilde{x} \text{ is a document stored in the workflow repository}\} \quad (2)$$

On the elements of both  $X$  and  $\tilde{X}$  read and write operations are allowed. The operation  $r(x)$  returns the actual value of the document  $x$ . The operation  $w(x, val)$  writes the value  $val$  to the document  $x$ . We define the preconditions and postconditions to the operation  $r(x)$  as follows:

$$r(x) \quad (3)$$

preconditions :  $\exists x$   
postconditions :  $\exists x : x = r(x)$



We define the preconditions and postconditions to the operation  $w(x, val)$  as follows:

$$\begin{aligned}
 w(x, val) & & (4) \\
 \text{preconditions : } & \text{none} \\
 \text{postconditions : } & \exists x : x = val
 \end{aligned}$$

Using these two operations we can pull a document  $x \in X$  from the external data source to the workflow repository or push a document  $\tilde{x} \in \tilde{X}$  from the workflow repository to an external data source. The operation  $op\_pull(\tilde{x}, x)$  is equivalent to  $w(\tilde{x}, r(x))$ . The operation  $op\_push(\tilde{x}, x)$  is equivalent to  $w(x, r(\tilde{x}))$ .

$$\begin{aligned}
 op\_pull(\tilde{x}, x) & \equiv w(\tilde{x}, r(x)) & (5) \\
 \text{preconditions : } & \exists x \in X \\
 \text{postconditions : } & \exists \tilde{x} \in \tilde{X} : \tilde{x} = r(x)
 \end{aligned}$$

$$\begin{aligned}
 op\_push(\tilde{x}, x) & \equiv w(x, r(\tilde{x})) & (6) \\
 \text{preconditions : } & \exists \tilde{x} \in \tilde{X} \\
 \text{postconditions : } & \exists x \in X : x = r(\tilde{x})
 \end{aligned}$$

If one of either operations  $op\_pull(\tilde{x}, x)$  or  $op\_push(\tilde{x}, x)$  was performed, then we say that documents  $x \in X$  and  $\tilde{x} \in \tilde{X}$  are in the copy relation  $K$ . The document  $x$  is called the original of  $\tilde{x}$  and the document  $\tilde{x}$  is called a copy of  $x$ :

$$(x, \tilde{x}) \in K, K \subseteq X \times \tilde{X} \quad (7)$$

The copy relation  $K$  has the following properties:

$$\forall \tilde{x} \in \tilde{X} : \tilde{x} \text{ is a copy of at most one } x \in X \quad (8)$$

$$\forall x \in X : x \text{ is the original of none or many } \tilde{x} \in \tilde{X} \quad (9)$$

### 3.2 WDL-X Operations on Documents

The WDL-X commands and operations mentioned in Sec. 2 can be now defined with the presented basic operations. Passing a variable to a simple activity in `or` mode, or testing a condition on a variable is equivalent to a read operation on a document bound to the variable. Receiving a variable from an activity in `or` mode is equivalent to a write operation on a document bound to the variable. If the variable was not bound to any document, a new document is created and bound to the variable. The exact semantics of these WDL-X operations depends on a chosen policy. In each policy the WDL-X command `openDocument` will create a copy of an external document in the workflow repository and will bind a variable to a newly created document. The refresh policies influence WDL-X read and `forceRefresh` operations as presented in Tab. 1. The push policies influence WDL-X write and `forcePush` operations as presented in Tab. 2.

**Table 1.** Semantics of WDL-X operations in the refresh policies

WDL-X operations	Basic operations		
	<b>refresh</b>	<b>refreshOnDemand</b>	<b>doNotRefresh</b>
<b>openDocument</b> (var, x)	$op\_pull(\tilde{x}, x)$ variable <b>var</b> will be bound to a new copy $\tilde{x}$ of $x$		
<b>r</b> ( $\tilde{x}$ )	if $\exists x \in X : (x, \tilde{x}) \in K$ then $op\_pull(\tilde{x}, x)$ $r(\tilde{x})$ else raise error	$r(\tilde{x})$	$r(\tilde{x})$
<b>forceRefresh</b> ( $\tilde{x}$ )	not allowed	if $\exists x \in X : (x, \tilde{x}) \in K$ then $op\_pull(\tilde{x}, x)$ else raise error	not allowed
<b>w</b> ( $\tilde{x}, val$ )	not influenced by these policies		
<b>forcePush</b> ( $\tilde{x}$ )			

**Table 2.** Semantics of WDL-X operations in the push policies

WDL-X operations	Basic operations		
	<b>immediatePush</b>	<b>pushOnDemand</b>	<b>doNotPush</b>
<b>openDocument</b> (var, x)	$op\_pull(\tilde{x}, x)$ variable <b>var</b> will be bound to a new copy $\tilde{x}$ of $x$		
<b>r</b> ( $\tilde{x}$ )	not influenced by these policies		
<b>forceRefresh</b> ( $\tilde{x}$ )			
<b>w</b> ( $\tilde{x}, val$ )	$w(\tilde{x}, val)$ $op\_push(\tilde{x}, x)$	$w(\tilde{x}, val)$	$w(\tilde{x}, val)$
<b>forcePush</b> ( $\tilde{x}$ )	not allowed	$op\_push(\tilde{x}, x)$	not allowed

The policies **refreshOnDemand** and **pushOnDemand** imply that in a WDL-X script there should be at least one **forceRefresh** and **forcePush** command respectively. Otherwise the policies degenerate to **doNotRefresh** and **doNotPush** respectively. Both problems are detected during the compile time and warnings are generated.

### 3.3 Combinations of Refresh and Push Policies

A workflow designer may define a different combination of refresh and push policies for each document copied into the workflow repository. In this section we shortly discuss each of the nine possible combinations. Each policy combination has to be analyzed separately for the documents copied into the workflow repository with the **openDocument** command and for the documents created first locally in the workflow repository.

1. In the policy combination **refresh immediatePush** documents copied into the workflow repository will be refreshed before each read operation and pushed after each write operation. Documents created in the workflow reposi-

tory will be pushed immediately after the first write operation and afterwards the situation is the same as for the copied documents.

2. In the policy combination **refresh pushOnDemand** documents copied into the workflow repository from external sources will be refreshed before each read operation. Local modifications of the copy in the workflow repository will be pushed only on an explicit **forcePush** command in a WDL-X script. A workflow designer has to be aware that, if in a sequence of WDL-X operations there is a read operation between a write operation and the **forcePush**, then the effect of this write operation will be overwritten by the refresh policy. If a new document was created in the workflow repository then this policy combination will lead to an error, when before a read operation the refresh policy tries to pull the document, which has not yet been pushed by the **forcePush** command.
3. In the policy combination **refresh doNotPush** documents copied into the workflow repository from external sources will be refreshed before each read access. Effects of all write operations will never be pushed and they will be always overwritten by the pull operations performed before subsequent reads. Any attempt to read a new document created locally in the workflow repository will always cause an error. Therefore, in this policy combination no write operations are permitted and the copy is . . . . .
4. In the policy combination **refreshOnDemand immediatePush** any write operation on a document copied into the workflow repository from an external source will be pushed immediately. Documents created locally in the workflow repository will be pushed immediately after the first write operation and afterwards the situation is the same as for the copied documents. To refresh the copy a workflow designer has to use explicitly the **forceRefresh** command.
5. In the policy combination **refreshOnDemand pushOnDemand** a workflow designer has full control by specifying when to pull and push a document copied into a workflow repository by using explicitly **forceRefresh** and **forcePush** commands respectively. The designer has to be aware that a new document created locally in the workflow repository has to be pushed before it can be pulled (i.e. **forcePush** has to precede **forceRefresh**). Otherwise the **forceRefresh** command will cause an error.
6. In the policy combination **refreshOnDemand doNotPush** any write operation on a document copied into the workflow repository from an external source will never be pushed and a workflow designer has to be aware that the **forceRefresh** command will always overwrite the effects of a preceding write operation. Any attempt to use **forceRefresh** on a document created locally in the workflow repository will always cause an error. Therefore, for all documents created only locally in the workflow repository this policy combination degenerates to the policy combination **doNotRefresh doNotPush**. Such a situation can be detected during the compile time, because in the WDL-X script there will be no **openDocument** command for the documents local to the workflow repository.

7. In the policy combination `doNotRefresh immediatePush` a document copied into the workflow repository from an external data source is pulled only once after the WDL-X command `openDocument`. A document created locally in the workflow repository will be immediately pushed. All write operations are immediately pushed. In this policy combination a workflow instance works on a local copy and externalizes all modifications made to this copy.
8. The policy combination `doNotRefresh pushOnDemand` is similar to the policy combination `doNotRefresh immediatePush` with the difference that to push modifications made to a document in the workflow repository a workflow designer has to explicitly use the command `forcePush`. In this policy combination a workflow instance works on a local copy, which is externalized only on demand.
9. In the policy combination `doNotRefresh doNotPush` a document copied into the workflow repository from an external data source is pulled only once after the WDL-X command `openDocument`. Afterwards the copy is never synchronized. A document created only in the workflow repository is never pushed and always stays a local document.

## 4 Example

We illustrate our approach with a simplified example of a credit application processing workflow. The workflow graph representing the workflow definition is presented in Fig. 2. The rectangles represent activities. Each activity has a name and an assigned agent specified in brackets underneath the name. Arrows between activities represent the control flow. A circle with a question mark inside and outgoing arrows represents an exclusive choice, whereas a circle with incoming arrows represents a simple merge. The WDL-X definition of the workflow is presented in Fig. 3.

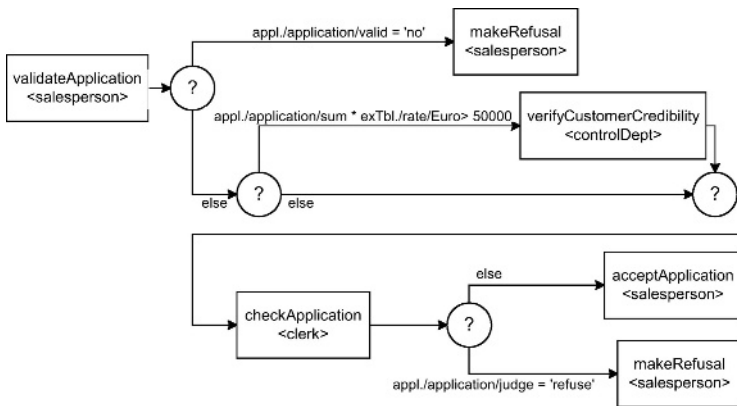


Fig. 2. Workflow graph of a credit application processing

```

1 process processCredit(IN appl : creditApplicationType)
2 documents customerData : customerType
3           accessedBy customerDbPlugIn refresh pushOnDemand,
4           exTbl : exchangeRateTableType,
5           accessedBy centralBankPlugIn refresh doNotPush;
6 begin
7   salesperson validateApplication(appl);
8   if (appl./application/valid/text()='no')
9     salesperson makeRefusal(appl);
10  else
11    openDocument(customerData, appl./application/customer/@id);
12    openDocument(exTbl, appl./application/sum/@currency);
13    if (appl./application/sum/text() * exTbl./rate/Euro/text() > 50000)
14      controlDept verifyCustomerCredibility(customerData);
15      forcePush(customerData);
16    endif;
17    clerk checkApplication(appl,customerData);
18    if (appl./application/judge/text()='refuse')
19      salesperson makeRefusal(appl);
20    else
21      salesperson acceptApplication(appl,customerData);
22    endif;
23  endif;
24 end;

```

**Fig. 3.** WDL-X script definition of a credit application processing

A credit application is passed to the process as an input parameter (line 1). A salesperson validates the application for correctness. The applications which are not valid are rejected. A sum of the credit in a valid application may be given in one of several currencies (e.g. USD, JPY, GBP). If the sum of the credit recalculated into Euro exceeds a value of 50 000 EUR, then the control department of the credit institution has to check the credibility of the customer. In any case valid applications have to be checked by a clerk. According to his / her judgment the salesperson accepts or rejects the application.

In the process apart from the credit application are used two additional documents: customer data (line 2) and the exchange rate table for a currency of the credit application (line 4). Both documents are stored locally in the workflow repository as copies of correspondent documents received from external data sources by data access plug-ins. According to policies defined by a workflow designer the customer data should be refreshed before each access and local changes should be pushed only after an explicit `forcePush` command, and the exchange rate table should be refreshed before each read and never pushed (lines 3 and 5).

The document describing customer data is copied from the external database according to the customer's id given in the application (line 11) and the exchange rate table is copied for the currency describing the credit sum (line 12). The exchange table is automatically refreshed before recalculating into Euro and checking the credit sum (line 13). Checking of the customer's credibility (line 14) may change the customer data in a significant manner. These changes are

therefore pushed to the original data source (line 15). Changes possibly made to the customer data by any other activity are never pushed and always overwritten from the original source before each read access in the workflow, e.g. before checking the application (line 17) and accepting the application (line 21). No modifications to the exchange rate table are allowed.

The example illustrates the versatility of our simple yet powerful mechanism. Our synchronization policies can model large spectrum of requirements ranging from fully synchronized replicas, through read only data to data local just to the workflow repository. In the example on the one hand the policy combination `refresh pushOnDemand` gives the workflow designer a control when to externalize updates made to the local copy of data. On the other hand the policy combination `refresh doNotPush` allows to model read only data. The former case allows to push to the original source and to take into account in the further workflow processing only important and verified data modifications. This is particularly important if activities and subprocesses are reused in different workflow definitions. If a subprocess reused in a workflow definition modifies the data, this modification can be simple discarded by the refresh policy or preserved by the `forcePush` command. The latter case describes the situation when in a workflow are used data, which are completely managed outside the context of an active workflow instance. These data cannot be changed by the workflow instance, nevertheless, the control flow of the workflow depends on them. Such read only data are currency exchange rate (as in our example), a marital status of a person, credit card validity, position of an employee, tax rates, metal and oil prices and many more.

## 5 Related Work

Data aspects in workflow systems did not yet receive the same attention as process aspects. There were even questions whether workflows had lost sight of the dataflow [3]. The Workflow Management Coalition (WfMC) in its glossary [15] defines three classes of data in workflows. *Workflow relevant data* are managed by a WfMS and describing workflow execution are not in scope of our interest here. *Application data* are managed by the applications supporting the process instance and generally are never seen by the WfMS. *Workflow control data* are used by the WfMS to determine the state transitions of a workflow (e.g. transition conditions). In the traditional WfMSs workflow relevant data must be stored in the workflow repository and the application data are beyond the WfMS. In [16] the WfMC defined environmental data, as data which may be accessed by workflow activities or used by the WfMS in the evaluation of conditional expressions in the same way as workflow relevant data. Unfortunately the WfMC does not specify in [16] anything else in this matter.

The analysis of commercial WfMSs shows that most of the activity programming is related to updating application databases [1]. Some products provide constructs for accessing external data sources. Typically these are specific elements that can be included in a process definition [11]. For example Staffware

provides Automatic Steps and script commands which enable specific items of data to be requested from external systems [13]. In @enterprise [8] to each activity can be attached Java code, which e.g. can fetch external data. In many systems (e.g. MQWorkflow [9]) access to external data occurs within individual simple activity implementation.

On the contrary we present a complete mechanism for accessing the external data by the WfMS. The data from the external systems may be used to control the flow of a workflow or copied into the workflow repository according to the policies defined by a workflow designer in a workflow definition.

The project Exotica/FMQM was extended to incorporate data management capabilities into the WfMSs [2]. Its architecture was based on a fully distributed workflow engine for control flow, and a set of loosely synchronized replicated databases which provided the common distributed repository for all the sites participating in the execution of a process. In this approach only control data were replicated in the distributed repository, whereas we provide the WfMS with a mechanism for copying and synchronizing data between external systems and the workflow repository.

Replication is a well known problem both in distributed systems and in databases [10, 12, 14]. In these domains data are usually replicated to provide better performance and data availability. Our proposal is specific to the workflow environment and has to deal with different requirements and problems. Mere replication management lacks the flexibility of more sophisticated synchronization policies. But according to the terminology in this domain in our case we can say that we have a multimaster replication. Data are stored in an external source and in the workflow repository and can be accessed and modified independently in both of them. Both copies are synchronized in a lazy manner. For simplicity we consider only a state transfer between copies. The main difference to the other work is that the copy conflicts are solved according to the policy defined by the workflow designer (e.g. the policy combination `refresh doNotPush` gives superiority to the external data source).

A more general approach to change propagation in heterogeneous information systems was presented in [4]. Dependencies between data objects stored in different information systems were managed by a separate system called Propagation Manager. This approach used wrappers to connect to external systems and allowed to specify scripts with data transformation definitions. In our proposal the WfMS decides when to propagate changes according to synchronization policies and a workflow definition. External data sources are accessed by specialized wrappers called data access plug-ins, which also provide the required data transformations.

## 6 Conclusions

This work contributes to a better handling of data in workflow management systems. We argue that data should get more attention in workflow systems and that making data access explicit rather than hiding it in activities has several

advantages, in particular, it improves understandability, maintainability and auditability of workflow definitions. With suitable abstractions and policy based synchronization workflow development will also be improved significantly.

We offer a simple and transparent way for accessing external data and thus, in particular, make the relationship between external data and workflow decision data more visible. We offer a transparent and manageable mechanism for synchronizing copies in the workflow repository with the external data with a clear semantics of operations on data copied into the workflow repository. The synchronization is policy driven relieving the workflow developer from low level programming details. Overall, the concepts introduced in WDL-X are intended to simplify workflow definition and maintenance.

A workflow designer can choose whether to access environmental data in external systems via data access plug-ins directly or to copy these data into the workflow repository. In the latter case he has full flexibility in choosing the synchronization policy. The mechanisms described in this paper builds a solid foundation for further support of workflow definition languages where synchronization specification is more automated by correctness specifications and analysis of workflow definitions.

## References

1. Martin Ader. Workflow and business process management comparative study. volume 2. Technical report, Workflow & Groupware Strategies, June 2003.
2. Gustavo Alonso, Berthold Reinwald, and C. Mohan. Distributed data management in workflow environments. In *Proceedings of the 7th International Workshop on Research Issues in Data Engineering (RIDE '97) High Performance Database Management for Large-Scale Applications*, page 82. IEEE Computer Society, 1997.
3. Christoph Bussler. Has workflow lost sight of dataflow?, 1999. High Performance Transaction System Workshop 1999.
4. Carmen Constantinescu, Uwe Heinkel, Ralf Rantza, and Bernhard Mitschang. A system for data change propagation in heterogeneous information systems. In *Proceedings of the International Conference on Enterprise Information Systems (ICEIS)*, volume I, pages 51–59, Ciudad Real, Spain, April 2002.
5. J. Eder, H. Groiss, W. Liebhart: *The Workflow Management System Panta Rhei*. In: A. Dogac, L. Kalinichenko, T. Öszu, A. Sheth (Eds.): *Workflow Management Systems and Interoperability*, Springer-Verlag 1998
6. Johann Eder and Marek Lehmann. Uniform access to data in workflows. In Kurt Bauknecht, Martin Bichler, and Birgit Pröll, editors, *Proceedings of the 5th International Conference on E-Commerce and Web Technologies, EC-Web 2004*, number 3182 in LNCS, pages 66–75, Zaragoza, Spain, August/September 2004. Springer-Verlag.
7. Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: from process modeling to workflow automation infrastructure. *Distrib. Parallel Databases*, 3(2):119–153, 1995.
8. Groiss Informatics GmbH `@enterprise` Documentation available at: <http://www.groiss.com>.
9. IBM Corporation. IBM WebSphere MQWorkflow Concepts and Architecture. GH12-6285, 2003



10. Abdelsalam A. Helal, Abdelsalam A. Heddaya, and Bharat B. Bhargava. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, 1996.
11. Nick Russell, Arthur H. M. ter Hofstede, David Edmond, and W.M.P. van der Aalst. Workflow data patterns. Technical Report FIT-TR-2004-01, Queensland University of Technology, Brisbane, Australia, April 2004.
12. Yasushi Saito and Marc Shapiro. Optimistic replication. Technical Report MSR-TR-2003-60, Microsoft Research, October 2003.
13. Staffware plc. *Staffware Technical Overview. Issue 1*, October 2001.
14. Matthias Wiesmann, André Schiper, Fernando Pedone, Bettina Kemme, and Gustavo Alonso. Understanding replication in databases and distributed systems. In *Proceedings of the The 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, page 464. IEEE Computer Society, 2000.
15. Workflow Management Coalition. *Workflow Management Coalition Terminology & Glossary*, 3.0 edition, February 1999.
16. Workflow Management Coalition. *Workflow Process Definition Interface - XML Process Definition Language (XPDL)*, wfmc-tc-1025 edition, 2002.

# Understanding the Requirements on Modelling Techniques

S.J.B.A. Hoppenbrouwers, H.A. Proper, and Th.P. van der Weide

Institute for Computing and Information Sciences, Radboud University Nijmegen,  
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, EU  
{S.Hoppenbrouwers, Th.P.vanderWeide, E.Proper}@cs.ru.nl

**Abstract.** The focus of this paper is *not* on the requirements of an information system to be developed, but rather on the requirements that apply to the *modelling techniques* used during information system development. We claim that in past and present, many information systems modelling techniques have been developed without a proper understanding of the requirements that follow from the development processes in which these techniques are to be used. This paper provides a progress report on our research efforts to obtain a fundamental understanding of the requirements mentioned. We discuss the underlying research issues, the research approach we use, the way of thinking (*weltanschauung*) that will be employed in finding the answers, and some first results.

## 1 Introduction

In past and present, many information systems modelling techniques have been, and are being, developed [1, 2, 3, 4]. With the term *modelling technique* we (roughly) refer to the combination of a modelling language/notation and procedures/guidelines for the creation of models. This definition is in line with definitions that can be found in e.g. [3, 4, 5]. The authors of this paper have themselves contributed their fair share of modelling techniques [6, 7, 8, 9, 10, 11]. The plethora of modelling techniques that is available to developers of information system has, in the past, already been referred to as “a jungle” [3]. This jungle leaves developers of information systems with the burden of selecting modelling techniques that are apt for the modelling tasks at hand.

The UML [12] aims to provide a standardisation of some of the diagramming techniques used. However, to a large extent it is still up to the modelers to choose the right diagramming technique for a given task, and the best way to apply it to this task. What is more, there are bound to be modelling tasks for which none of the UML standard techniques suffice. There is no silver bullet [13] for modelling techniques.

This leads us to the key questions with which the research, as reported in this paper, is concerned:

— How can we understand the requirements on modelling techniques?

In our view, selecting and applying modelling techniques depends strongly on the goals of the tasks that are to be executed in the development process. These goals dictate the requirements that should be set for the modelling technique and their usage. In this paper, we will show how these requirements depend on a multitude of factors. In doing so, we will base ourselves both on theoretical considerations and on input from interviews with practitioners.

In finding answers to the questions raised above, we employ the *diagnostic and therapeutic approach* [14]. This entails that our work will progress (evolutionary) through two major stages (taken from [15]):

**Diagnostic stage** – The diagnostic stage is the first stage in the diagnostic and therapeutic approach. It is characterized by a focus on understanding the current situation and the requirements for the development process. This stage is characterized by a focus on understanding the current situation and the requirements for the development process.

**Therapeutic stage** – The therapeutic stage is the second stage in the diagnostic and therapeutic approach. It is characterized by a focus on addressing the requirements identified in the diagnostic stage. This stage is characterized by a focus on addressing the requirements identified in the diagnostic stage.

We are currently in transition from the diagnostic stage to the therapeutic stage. In the execution of the diagnostic stage, we have used a three-pronged approach, which has also inspired the structure of the remainder of this article:

**Articulate way of thinking** – This sub-stage refers to the way of thinking (or *mindset*) concerning information system development we will embrace in seeking the answers to the questions raised above. The current status of this sub-stage is discussed in section 2.

**Define conceptual framework** – The focus of this sub-stage is on a further refinement and concretisation of the way of thinking in terms of a conceptual framework. This framework is needed to position and “code” the empirical results that will follow from the execution of the next stage (the therapeutic stage). The current status of this sub-stage is addressed in section 3.

**Initial findings** – The elaboration of the conceptual framework took place in conjunction with a number of interviews with experienced modelers (in particular enterprise architects)<sup>1</sup>. These interviews and discussions already produced some results that provide a more practical perspective on the theoretical framework. These results are presented in section 4.

## 2 Communication-Driven Knowledge Transformation

In this section we discuss our fundamental way of thinking with regard to system development. It provides a frame of thought against which one can better understand the (communicative) requirements posed on modelling techniques.

<sup>1</sup> This part of the research was conducted within the context of the ArchiMate project (<http://archimate.telin.nl>), a research projects that aims to provide concepts and techniques to support enterprise architects in the visualisation, communication and analysis of integrated architectures.

## 2.1 Communication-Driven

Key to our view on the utility of modelling techniques is their role as a means of communication in system development. In the past we have already taken a communication-driven perspective on modelling activities in information system development [11, 17, 18, 19, 20], as well as on the act of system development itself [21]. We are certainly not alone in doing so [22, 23].

To understand the role of modelling techniques in system development, we have extended our communicative perspective to cater for the fact that the communication taking place during system development leads to the creation and dissemination of *system descriptions*. In essence, we regard system development as a communication-driven *knowledge transformation* process whereby conversations are used to share and create knowledge pertaining to the system being developed as well as the development process as such. The notion of *system description* should be interpreted here in the broadest sense, ranging from a single person producing a model (description), via one-on-one design/elicitation sessions, to workshops with several stakeholders, and even the broad dissemination of definitive system designs.

Our aim of viewing information system development as a knowledge transformation process is to use this perspective on system development to better understand and articulate the requirements that should be set for modelling techniques. From this perspective, modelling techniques should be regarded as a means (a language) to an end (system development), not unlike a functional perspective (*functional programming*) on language [24].

## 2.2 Development Community

Given our focus on communication, it is important to identify the *actors* and *objects* that could play a role in the communication that takes place during the system development process. The actors are likely to have some stake with regards to the system being developed. Examples of such actors are: problem owners, prospective actors in the future system (such as the future ‘users’ of the system), domain experts, sponsors, architects, engineers, business analysts. The actors, however, are not the only items playing an important role in system development. In addition, consider a number of objects: the many different documents, models, forms, etc., that *represent* bits and pieces of knowledge pertaining to the system that is being developed. Actors and objects combined, and the different roles they can play, is what we shall refer to as a *development community*.

The *actors* in a system development community will (typically as a consequence of their personal *roles* and *interests*) have some specific interests with regard to the system being developed. This interest implies a sub-interest regarding (the contents of) the system descriptions that are communicated within the community. This interest is, in line with [25], referred to as the *concern* of a stakeholder. Some examples of concerns are:

- The current situation concerning the computerized support of a business process.
- The requirements of a specific stakeholder with regard to the desired situation.

- The improvements/benefits which a new system may bring to a pre-existing situation in relation to the cost of acquiring the system.

### 2.3 System Development Knowledge

The system development community harbours knowledge about the system being developed. To be more precise, the members of the system development community can be regarded as *knowledge carriers* harbouring knowledge pertaining to (their view on) a sub-domain within the system being developed (and/or its development process). In this vein, the communication occurring within a system development community essentially aims to create, further, and disseminate this knowledge. Importantly, the actual knowledge can pertain to the system being developed, as well as the development process as such. In the next section, we will provide a more elaborate discussion on the kinds of knowledge that may (have to) be communicated.

Depending on the concerns of a stakeholder, she will be interested in different knowledge topics pertaining to the system being developed. For example: a financial controller will be interested in an investment perspective on the overall scope of a future system, a designer will be interested in all aspects of the design chain from different perspectives, etc.

### 2.4 Transformations of Knowledge

During the development of a system, the knowledge about the system and its development will evolve. New insights emerge, designs are created, views are shared, opinions are formed, design decisions made, etc. Consequently, the knowledge as it is present in a development community can be seen to evolve through a number of *knowledge states*. At present, we identify two dimensions for the knowledge states of the development community: (1) level of sharing, and (2) level of explicitness.

Knowledge needs to be *introduced* into the community first, either by creating the knowledge internally or importing it from outside of the community. Once the knowledge has been introduced to a community, it can be *shared* among different knowledge carriers. Sharing knowledge between different knowledge carriers may progress through a number of stages. Of the two kinds of knowledge carriers (objects and actors) in a development community, only the actors are “allowed” to cast judgement on the level of sharing between two knowledge carriers. We actually distinguish three major stages of knowledge sharing:

- Aware** – An carrier may become “aware” of (possible) knowledge by way of the sharing by another carrier (possibly from outside the community), or by creating it themselves.
- Agreed** – When shared, carriers can make up their own “minds” about the shared knowledge, and decide whether or not to *commit* to the knowledge shared.
- Committed** – Carriers who “agree” to a specific knowledge topic may decide to actually commit to this knowledge. In other words, they may decide to adopt their future behaviour in accordance to this knowledge.

There is no way to objectively and absolutely determine the levels of awareness, agreement, and commitment of a given set of knowledge carriers. It is in the eyes of the beholder.

The actual knowledge that is harboured by a knowledge carrier can also not be taken into account since the knowledge that is available from/on/in a knowledge carrier is subjective and context-dependent by nature [26]. The harbouring of a knowledge topic by some knowledge carrier may occur at different levels of formality, completeness, executability, etc. In the field of knowledge management, a key distinction is made between *explicit* and *implicit* knowledge [27]. *Explicit knowledge* refers to knowledge that can be externalised in terms of some representation. In representation of knowledge, we refer to the process of encoding knowledge in terms of some language on some medium. Our focus is on the communication of system development knowledge by way of *explicit* representations. In other words, *explicit knowledge*, where the representations pertain to an existing or future system; its design, the development process by which it was/is to be created, the underlying considerations, etc.

### 3 Conceptual Framework

Following the general way of thinking as discussed in the previous section, in this section we present a conceptual framework. This framework will be used in the further development of our theories based on its use in practical settings.

#### 3.1 System Development Knowledge

We start by briefly exploring the kinds of knowledge that are relevant to a system and its development, in other words: the knowledge topics that can be discerned. During system development, members of the system development community will create and exchange knowledge pertaining to different topics. A first distinction can be made between:

**Target domain** – Knowledge pertaining to the *system* being developed.

**Project domain** – Knowledge about the *development process* that brings forth the system.

We have borrowed the terms *target domain* and *project domain* from the Information Services Procurement Library (ISPL) [28]. For both of these knowledge domains, further refinements can be made with regards to the possible topics. One can identify the following additional characterizations:

**Perspective** – Artifacts, such as systems, can be considered from different perspectives. Some examples are: (1) Business, application, and infrastructure aspects of a (computerized) information system; (2) Social, symbolical, and physical aspects of a system; (3) Process, information, actors, and technology featuring in a system.

**Scope** – Given a domain, such as a system or a development project, several scopes can be identified when approaching the domain. Some examples are: (1) enterprise wide; (2) department specific; (3) task specific.

**Design chain** – When considering the design of some artifact, a distinction can be made between: (1) the **purpose** for which an artifact is needed; (2) the **functionality** which the artifact should provide to its environment; (3) the **design** of the artifact, i.e. . . . it should realize the functionality; (4) the **quality** of the artifact, i.e. . . . it should do so; (5) the **cost** at which it will/may do so, and at which it may be constructed.

Based on these distinctions, knowledge topics can be characterized in terms of their focus on, for example, functionality or quality in . . . , or their focus on bridging the gaps between purpose, functionality and design in terms of the underlying . . .

**Historical perspective** – Given an artefact with a design, one may consider different versions of this design over time. One could, for example, make a distinction between a strategic (5-10 years), a tactical (1-5 years), and an operational perspective (now).

**Abstraction level** – When considering a domain, one may do so at several levels of abstraction. Various forms of abstraction can be distinguished, for example type-instance, generalisation, is-a-kind-of, encapsulation, and the hiding/encapsulation of implementation details.

In general, each of the above characterizations of knowledge topics applies to both target and project domains. As mentioned before, depending on their concerns, stakeholders may be interested in different knowledge topics.

### 3.2 Explicitness of Knowledge

Given our focus on system development, a more precise classification can be made with regard to the . . . as mentioned in section 3.2. Based on [28, 29], the following dimensions of explicitness for representations of system development knowledge (pertaining to both target domain and project domain knowledge) can be identified:

**Level of formality** – The degree of formality indicates the type of representation language used. Such a language could be formal, in other words a language with an underlying well-defined semantics in some mathematical domain, or it could be informal –not mathematically underpinned; typically texts in natural language, graphical illustrations, animations, etc.

**Level of quantifiability** – Different aspects of the designed artefact, be it (part of) the target or the project domain, may be quantified. Quantification may be expressed in terms of volume, capacity, workload, effort, resource, usage, time, duration, frequency, etc.

**Level of executability** – The represented knowledge may, where it concerns artefacts with operational behaviour, be explicit enough so as to allow for execution. This execution may take the form of a simulation, a prototype, generated animations, or even fully operational behaviour based on executable specifications.

**Level of comprehensibility** – The knowledge representation may not be comprehensible to the intended audience. Tuning the required level of comprehensibility of the representation, in particular the representation language used, is crucial for effective communication. The representation language may offer special constructs to increase comprehension, such as stepwise refinements, grouping/clustering of topically related items/statements, etc.

**Level of completeness** – The knowledge representation may be complete, incomplete, or overcomplete with regard to the knowledge topic (see previous subsection) it intends to cover.

### 3.3 Conversation Strategies

The knowledge transformations as discussed in section 2.4 are brought about by conversations. The scope of these conversations may range from ‘atomic’ actions involving a small number of actors, via discussions and workgroups, to the development process as a whole. This has been illustrated informally in figure 1.

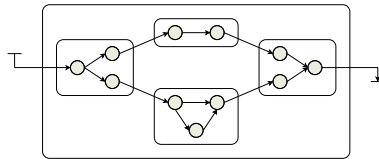


Fig. 1. Example sequence of conversations

Each conversation is presumed to have some  $g$ : a knowledge state which the conversation aims to achieve (or maintain). This knowledge state can best be regarded as a multi-dimensional vector, positioning: (1) the knowledge topic (see section 3.1); (2) the level of explicitness of the knowledge (see section 3.2); (3) the level of sharing (see section 2.4).

In achieving a knowledge goal, a conversation will follow a  $s \rightarrow g$ . Such a strategy is needed to achieve the goal of the conversation, starting out from the current state:

**Knowledge goal** – The knowledge goal; a desired knowledge state which the conversation will aim to achieve/contribute towards.

**Initial state** – The initial knowledge state as it holds at the start of the conversation.

Conversations take place in some situation in which resources may or may not be available for execution of the conversation. A conversation situation may be characterised further in terms of  $s$  [28]. We identify three classes of situational factors:



**Availability of resources** – Refers to the availability of resources that can be used in a conversation. The availability of resources can be refined to more specific factor such as: time for execution, actors present, intellectual capacities required from the actors, and financial means.

**Complexity** – The resources needed for the conversation, the knowledge being conversed about, etc., will exhibit a certain level of complexity. This complexity also influences the conversation strategy to be followed. Examples of such complexity factors (inspired by [28]) are: heterogeneity of actors involved, quantity of actors involved, complexity of technology used, complexity of knowledge being conversed about, and size of the gap between the initial knowledge state and the desired knowledge state.

**Uncertainty** – In determining a conversation strategy fit for a given situation, assumptions will have to be made about the knowledge goal, the initial state, the availability of resources, and the complexities of these factors. During the execution of a conversation, some assumptions may prove to be wrong. For example: the commitment of certain actors may be lower than anticipated (initial state); materials needed for a workshop may not be available on time (resources); during a requirements elicitation session it may emerge that the actors involved do not (yet) have enough knowledge about the future system and its impact to formulate/reflect on the requirements of the future system (initial state). Typical uncertainty factors could relate to: the knowledge goal, the initial state, the abilities of actors involved, the availability of resources, and the complexities as discussed above.

In formulating a conversation strategy, all of the above factors should be taken into account. A conversation strategy should typically cover at least the following elements:

**Execution plan** – As mentioned before, a conversation can be composed of sub-conversations. Each of these sub-conversations focusses on a sub-goal, but they all contribute towards the goal of the conversation as a whole. The execution plan of a (composed!) conversation consists of a set of sub-conversations, together with a planned execution order.

**Description languages** – The description languages to be used in the conversation(s).

**Media** – The kind of media to be used during the conversation(s).

**Cognitive mode** – The *modus operandi* refers to the way in which knowledge is processed/gathered by the collective of actors involved in a conversation. Typically, a distinction is made between an analytical and experimental approach.

**Social mode** – The *modus operandi* is the way in which the actors executing the system development process collaborate with the actors from the business domain. We distinguish between an expert-driven and a participatory approach.

**Communication mode** – A small number of basic patterns of communication can be distinguished, as covered by combinations of the some basic factors:

speaker-hearer ratio, requirements on hearer response, allowed time-lag, locality, and persistency. Combinations of these factors can be used to typify many different modes of communication, which can have a major impact on the resources required for communication and the likelihood a knowledge goal is achieved.

## 4 Guidelines for the Usage of Modelling Techniques

This section concerns guidelines that should help practitioners in selecting modelling techniques for communication tasks at hand. These guidelines are based on interviews with a number of architects. In general, the use of a modelling technique will pass through a number of phases. These phases are:

**Phase 1 – Scoping:** Select (an) appropriate modelling technique, select the (sub)domain that needs to be represented/modelled in terms of (a) model(s), and determine the constraints that apply to the domain being modelled.

**Phase 2 – Creation of models:** Create/select the actual content of the models. This can pertain to the selection of a part of a larger (pre-existing) model, or the creation/refinement of a part of a model.

**Phase 3 – Validation:** Validate the resulting model. Do the stakeholders agree to the fact the model is a correct representation of the actual/intended situation?

**Phase 4 – Obtaining commitment:** If agreement has been reached among the key stakeholders involved, the next step will be to create a commitment for the results. In other words, do the stakeholders commit themselves to the (potential) impact of what is described by the model?

**Phase 5 – Informing:** Inform other stakeholders of the results. These stakeholders will be those members of the development community whose explicit commitment has, in a previous decision, been considered not to be crucial.

Note that these phases will not necessarily be executed in a linear order. Practical circumstances usually dictate a more evolutionary approach. Any modelling techniques is to be used in activities from each of the above phases. The guidelines resulting from the interviews are categorised according to these phases, and are discussed below.

### 4.1 Scoping

The importance of focussing on the concerns of stakeholders and the extent to which a specific modelling technique addresses these concerns, was confirmed by the outcomes of the interviews:

- ... ( ... ) ...
- ...
- ...

...the selection of modelling techniques should be deliberate and based on rational considerations. What is more, architects stated that this decision, and its rationalisation, must be readily available for communication during the different phases:

The selection of modelling techniques should be deliberate and based on rational considerations. What is more, architects stated that this decision, and its rationalisation, must be readily available for communication during the different phases:

- ...the selection of modelling techniques should be deliberate and based on rational considerations. What is more, architects stated that this decision, and its rationalisation, must be readily available for communication during the different phases:
- ...the selection of modelling techniques should be deliberate and based on rational considerations. What is more, architects stated that this decision, and its rationalisation, must be readily available for communication during the different phases:
- ...the selection of modelling techniques should be deliberate and based on rational considerations. What is more, architects stated that this decision, and its rationalisation, must be readily available for communication during the different phases:

#### 4.2 Creation of Models

During the creation of a model, in particular when actual modelling (i.e. not elicitation etc.) is concerned, it is considered sensible to limit the number of participants in a conversation:

- ...the selection of modelling techniques should be deliberate and based on rational considerations. What is more, architects stated that this decision, and its rationalisation, must be readily available for communication during the different phases:

During the early stages of system design, it is often considered a bad thing to “think” in terms of “solutions”. According to some of the interviewed architects, however, it is sometimes defensible to let this “thinking in terms of examples” run its course, as long as the results are expressed at the correct level of abstraction:

- ...the selection of modelling techniques should be deliberate and based on rational considerations. What is more, architects stated that this decision, and its rationalisation, must be readily available for communication during the different phases:

The above observation is actually a concretisation of a more abstract observation:

- ...the selection of modelling techniques should be deliberate and based on rational considerations. What is more, architects stated that this decision, and its rationalisation, must be readily available for communication during the different phases:

The use of concrete examples is but a way to make the different potential scenarios more tangible.

The graphical notation that is part of a modelling technique should be approached flexibly when it comes to communicating with stakeholders (in particular non-technical ones):

- The graphical notation should be adapted to the needs of the stakeholders.
- The graphical notation should be adapted to the needs of the stakeholders.
- The graphical notation should be adapted to the needs of the stakeholders.
- The graphical notation should be adapted to the needs of the stakeholders.

Finally, during a modelling session, several things may come to the fore that will influence the further process. External events may occur that are a threat to the process as a whole:

- External events may occur that are a threat to the process as a whole.
- External events may occur that are a threat to the process as a whole.
- External events may occur that are a threat to the process as a whole.

### 4.3 Validation

In validation, a clear difference should be made between validation of content (qualitative validation, by modelers and experts) and validation in terms of commitment (by executives). Both are crucial, but very different. Obtaining (and validating) commitment is discussed in the next subsection.

Whether good mutual communication and understanding about a model is being reached is often a matter of intuition:

- Whether good mutual communication and understanding about a model is being reached is often a matter of intuition.
- Whether good mutual communication and understanding about a model is being reached is often a matter of intuition.

Validation is an activity that should be conducted in limited groups:

- Validation is an activity that should be conducted in limited groups.
- Validation is an activity that should be conducted in limited groups.

### 4.4 Obtaining Commitment

Obtaining commitment for a specific architectural design involves obtaining commitment for the... of this design on the future system and its evolution, as well as the ... needed to arrive at this future system. This means that the message that one needs to get across to the stakeholders involves:

- What are the major problems in the current situation?
- How bad are these problems (to the concerns and objectives of the stakeholders)?
- How will this improve in the new situation? (Benefits)
- At what costs will these improvements come?

When discussing costs and benefits with stakeholders, it is important to realize the following:

- ...
- ...

Selecting the stakeholders that should be involved when obtaining commitment is also of key importance. Involving the wrong stakeholders, or leaving out important ones, will have obvious repercussions. At the same time, selecting too large a group of stakeholders may make the process bog down.

- ...
- ... also ...
- ...

The architects interviewed also noted some potential pitfalls in obtaining commitment:

- ...
- ...
- ...

The latter point clearly confirms that the linear ordering of the “technique use phases” as provided at the start of this section should not be applied strictly.

### 4.5 Informing

Once commitment from the opinion leaders has been obtained, other stakeholders may be informed about future plans and their impact. In doing so, it still makes sense to concentrate on cost/benefit considerations when trying to “sell” the new system. Below, we have gathered some observations that apply to the informing phase. However, due to their general communicative nature, some of these observations are also applicable to the creation, validation, and commitment phases.

- ... their ...
- ...
- ...

## 5 Conclusion

We presented a progress report on one of our ongoing research efforts. We discussed our way of thinking regarding system development as a communication-driven knowledge transformation process, and refined this way of thinking in terms of a conceptual framework. Finally, some first results have been discussed. These results are guidelines based on interviews with (enterprise) architects, which were conducted as part of the “diagnostic” stage of the action research paradigm.

We are currently in the process of initiating the “therapeutic” stage of the action research paradigm. Our plan is to participate in selected activities in development processes taking place in large Dutch cooperations and/or governmental agencies.

## References

1. Bubenko, J.: Information System Methodologies - A Research View. In Olle, T., Sol, H., Verrijn-Stuart, A., eds.: Information Systems Design Methodologies: Improving the Practice. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU (1986) 289–318.
2. Avison, D., Wood-Harper, A.: Information Systems Development Research: An Exploration of Ideas in Practice. *The Computer Journal* **34** (1991) 98–112.
3. Avison, D.: Information Systems Development: Methodologies, Techniques and Tools. 2nd edn. McGraw-Hill, New York, New York, USA (1995). ISBN 0077092333

4. Bernus, P., Mertins, K., Schmidt, G., eds.: Handbook on Architectures of Information Systems. International Handbooks on Information Systems. Springer, Berlin, Germany, EU (1998). ISBN 3-540-64453-9
5. Olle, T., Hagelstein, J., Macdonald, I., Rolland, C., Sol, H., Assche, F.v., Verrijn-Stuart, A.: Information Systems Methodologies: A Framework for Understanding. Addison-Wesley, Reading, Massachusetts, USA (1988). ISBN 0-201-54443-1
6. Bommel, P.v., Hofstede, A.t., Weide, T.v.d.: Semantics and verification of object-role models. *Information Systems* **16** (1991) 471–495.
7. Hofstede, A.t., Weide, T.v.d.: Expressiveness in conceptual data modelling. *Data & Knowledge Engineering* **10** (1993) 65–100.
8. Bronts, G., Brouwer, S., Martens, C., Proper, H.: A Unifying Object Role Modelling Approach. *Information Systems* **20** (1995) 213–235.
9. Creasy, P., Proper, H.: A Generic Model for 3-Dimensional Conceptual Modelling. *Data & Knowledge Engineering* **20** (1996) 119–162.
10. Campbell, L., Halpin, T., Proper, H.: Conceptual Schemas with Abstractions – Making flat conceptual schemas more comprehensible. *Data & Knowledge Engineering* **20** (1996) 39–85.
11. Hoppenbrouwers, J., Vos, B.v.d., Hoppenbrouwers, S.: NI structures and conceptual modelling: Grammalizing for KISS. *Data & Knowledge Engineering* **23** (1997) 79–92.
12. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modelling Language User Guide. Addison-Wesley, Reading, Massachusetts, USA (1999). ISBN 0-201-57168-4
13. Brooks, R.: Studying programming behavior experimentally: The problems of proper methodology. *Communications of the ACM* **23** (1980) 207–213.
14. Avison, D., Lau, F., Meyers, M., Nielsen, P.: Action research. *Communications of the ACM* **42** (1999) 94–97.
15. Baskerville, R.: Investigating Information Systems with Action Research. *Communications of the Association for Information Systems* **2** (1999).
16. Blum, F.: Action research – a scientific approach? *Philosophy of Science* **22** (1955) 1–7.
17. Derksen, C., Frederiks, P., Weide, T.v.d.: Paraphrasing as a Technique to Support Object-Oriented Analysis. In Riet, R.v.d., Burg, J., Vos, A.v.d., eds.: Proceedings of the Second Workshop on Applications of Natural Language to Databases (NLDB'96), Amsterdam, The Netherlands (1996) 28–39.
18. Frederiks, P., Weide, T.v.d.: Information modeling: the process and the required competencies of its participants. In Meziane, F., Métais, E., eds.: 9th International Conference on Applications of Natural Language to Information Systems (NLDB 2004). Volume 3136 of Lecture Notes in Computer Science., Manchester, United Kingdom, EU, Springer-Verlag, Berlin, Germany, EU (2004) 123–134.
19. Bleeker, A., Proper, H., Hoppenbrouwers, S.: The role of concept management in system development – a practical and a theoretical perspective. In Grabis, J., Persson, A., Stirna, J., eds.: Forum proceedings of the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004), Riga, Latvia, EU, Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, EU (2004) 73–82. ISBN 9984-9767-0-X
20. Proper, H., Hoppenbrouwers, S.: Concept evolution in information system evolution. In Gravis, J., Persson, A., Stirna, J., eds.: Forum proceedings of the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004), Riga, Latvia, EU, Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, EU (2004) 63–72.

21. Veldhuijzen van Zanten, G., Hoppenbrouwers, S., Proper, H.: System Development as a Rational Communicative Process. *Journal of Systemics, Cybernetics and Informatics* **2** (2004). <http://www.iiisci.org/Journal/sci/pdfs/P492036.pdf>
22. Embley, D., Kurtz, B., Woodfield, S.: *Object-Oriented Systems Analysis – A model-driven approach*. Yourdon Press, Englewood Cliffs, New Jersey, USA (1992). ASIN 0136299733
23. Halpin, T.: *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Mateo, California, USA (2001). ISBN 1-55860-672-6
24. Cruse, A.: *Meaning in Language, an Introduction to Semantics and Pragmatics*. Oxford University Press, Oxford, United Kingdom, EU (2000). ISBN 0-198-70010-5
25. The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE: *Recommended Practice for Architectural Description of Software Intensive Systems*. Technical Report IEEE P1471-2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA (2000). ISBN 0-738-12518-0 <http://www.ieee.org>
26. Peirce, C.: *Volumes I and II – Principles of Philosophy and Elements of Logic*. *Collected Papers of C.S. Peirce*. Harvard University Press, Boston, Massachusetts, USA (1969). ISBN 0-674-13800-7
27. Nonaka, I., Takeuchi, H.: *The knowledge-creating company*. *Harvard Business Review* (1991) 97–130.
28. Franckson, M., Verhoef, T., eds.: *Managing Risks and Planning Deliveries*. *Information Services Procurement Library*. ten Hagen & Stam, Den Haag, The Netherlands (1999). ISBN 9076304831
29. Proper, H., ed.: *ISP for Large-scale Migrations*. *Information Services Procurement Library*. ten Hagen & Stam, Den Haag, The Netherlands, EU (2001). ISBN 9076304882



# A Process for Generating Fitness Measures

Anne Etien and Colette Rolland

Centre de Recherche en Informatique – University Paris 1,  
90 rue de Tolbiac, 75013 Paris, France  
{aetien, rolland}@univ-paris1.fr

**Abstract.** It is widely acknowledged that the system functionality captured in a system model has to match organisational requirements available in the business model. However, fitness measures are rarely integrated in design methodologies. The paper proposes a framework to ease the generation of fitness measures adapted to a given methodology in order to quantify to which extent there is fit between the business and the system. The framework comprises a generic level and a specific level. The former provides generic evaluation criteria and metrics expressed on the basis of business and system ontologies. The specific level is dealing with a specific set of metrics adapted to specific business and system models. The paper presents the process for generating a specific set of measures from the generic set, illustrates it with two specific models and shows how the use of the generated metrics can help in making design decisions in the development of a hotel room booking system.

## 1 Introduction

Fitting information systems (IS) to business processes (BP) that they support, is equally considered important by researchers and by professionals [1], [2]. Recent field surveys seem to demonstrate this importance. For example, a 2001 study conducted in 226 companies [3] clearly proves that alignment of IS to BP significantly improve business performance. Complementarily, Henderson and Venkatraman [4] show that the lack of fit of information systems to business strategies is the reason why business processes fail in providing the return on IT investments.

Researchers are interested by mechanisms to get the alignment done. Most IS development methodologies propose a step-wise process to ensure that the designed IS matches the business needs and strategies. Old methodologies such as the participative method [5], more recent object oriented methodologies [6] and the entire requirements engineering community [7] promote approaches based on goal models to capture the business strategies and on transition rules to operationalise goals into IS solutions. However, these methodologies do not provide means to evaluate if there is fit and to which extent.

P. Soffer [8] suggests that identification of unfit requires the application of a fit measurement method. Measurement is indeed a way to avoid a subjective evaluation of the degree of fit. We follow this line and our concern is to define a set of fit measures that could be easily incorporated in any existing methodology. This raises

two issues: (a) the definition of the concepts of fitness and fitness measurement and, (b) the production of fitness measures for a specific methodology.

In dealing with issue (a) we adopt the view of Regev [9], who defines the concept of fitness “as the correspondence between a set of components”. In the case of BP/IS fitness evaluation, this view implies a precise identification of the types of correspondence between components of the business model (BM) and the system model (SM). The measurement of BP/IS fit is therefore, based on the degree of correspondence between BM and SM components. We propose the use of fitness criteria and associated metrics to measure these.

To addressing issue (b) we base the process of producing fitness measures for a specific methodology on the framework shown in Fig. 1. The specific set of measures is derived from a generic set of measures. The former is based on correspondences established between components of a specific business model and a specific system model whereas the latter are associated to correspondences between generic constructs found in the Wand and Weber [10] and Soffer and Wand [11] ontologies, respectively. These two ontologies are adaptations of Bunge’s ontology [12], [13] which is largely recognized for its theoretical foundations. We believe that there are a number of advantages of proceeding in this way, (1) the generic measures are based on a solid theoretical ground provided by the Bunge’s ontology (2) the generic measures serve as a guide to define the specific ones: the latter is just a specialisation of the former, (3) the process of producing the specific measures is easier and less error prone and, (4) specific sets of fit measures are consistent with each other as they are generated from the same mould and this facilitates comparisons across methods.

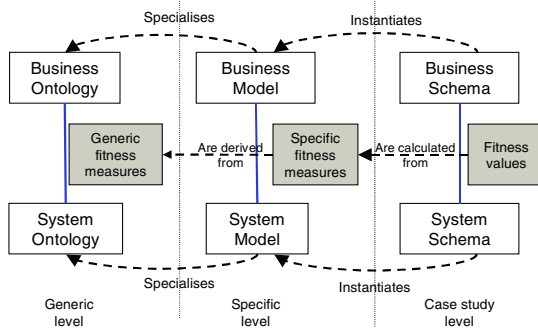


Fig. 1. Framework for generating specific fit measures

In this paper, we illustrate the use of our generic set of fitness measures to generate a specific set of measures. For this purpose, we propose a three steps process. In the two first steps, a correspondence between constructs of the specific business model (respectively, system model) and those of the business ontology (respectively, system ontology) is established. The third step consists in the specification of the generic metrics based on the correspondences identified in the two previous steps.

The rest of the paper is organized as follows. Section 2 provides an overview of the ontologies and of the generic set of fitness criteria and associated metrics. Section 3 presents the process to generating specific metrics and illustrate it with the MAP [14] and O\* [15] models. In section 4 the use of the specific metrics generated in section 3 is discussed in a case of a hotel room booking. Finally, conclusions are drawn in section 5.

## 2 Generic Level: Overview of the Fitness Measurement System

This section provides an overview of the generic set of criteria and metrics that we defined to measure to which extent there is fit between the software system and the business it supports. We first, recall the Soffer and Wand (SW) and Wand and Weber (WW) ontologies, which we use to represent system and business components, respectively. We then, present our view of the fit as the degree of correspondence between SW and WW elements computed by metrics. We finally sum up the fit criteria and associated metrics.

### 2.1 Ontologies Overview

The Soffer and Wand ontology (SW) [11] and the Wand and Weber (WW) [10] ontology are summed up in the meta-models of Fig. 2. Both ontologies are specialisations of the Bunge ontology [12] and therefore, share a number of constructs. The core concept is the one of *thing*. A thing has some *properties* that are perceived in terms of *attributes*. For a given thing, the set of values of all its attribute functions is called its *state*. Properties can change and therefore, things undertake state changes called *events*. There exist rules governing possible states and possible state changes called *state laws* and *transition laws*, respectively.

The SW ontology differs from the WW ontology by emphasizing the constructs of *goal* and *process*. A goal is defined as a set of *stable states* and a process as a sequence of *unstable states* leading to a goal.

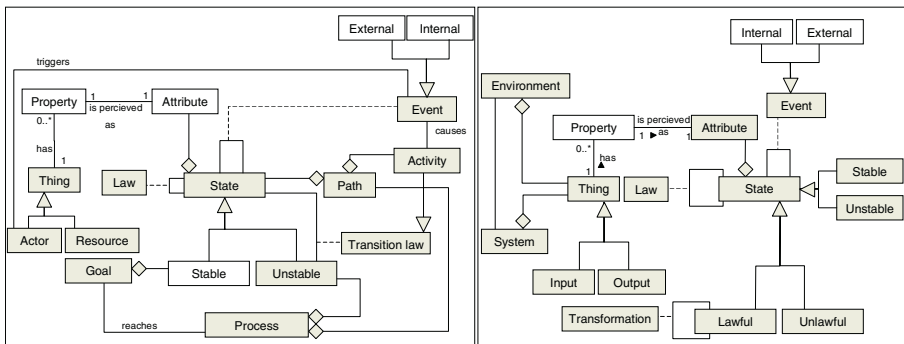


Fig. 2. Meta-model of the SW (left) and the Wand and Weber (right) ontologies

## 2.2 Correspondences Between SW and WW Constructs

In line with Regev [9], we view the fit between the system and the business as the degree of correspondence between SW and WW components. We found that two types of correspondence links were relevant, namely *maps* and *represents*. The former expresses equality between SW and WW identical constructs. The latter specifies that a WW construct has an impact on a SW construct. Thus, two constructs of different nature, for example a WW thing and a SW property or a WW event and a SW activity can be linked by a *represents* link.

A SW construct *X maps* ( $\mathcal{M}$ ) a construct *Y* of the same nature if there exists a function  $f$  such as the set of elements  $f(X)$  equals the set of elements  $f(Y)$

$$X \mathcal{M} Y \Leftrightarrow f(x) = f(y)$$

In the case of things for example,  $f(X)$  corresponds to the set of properties of the thing *X*. Between two states,  $f(X)$  corresponds to set of values of *X*. The *maps* link between a state *X* at the business level and a state *Y* at the system level implies that (i) the set of values of all the attribute functions of one thing equals the set of values of all the attribute functions of the other thing but also that (ii) these two sets of attribute functions are identical and thus that the two things *X* and *Y* *map* each other.

A WW construct *represents* ( $\mathfrak{R}$ ) a SW construct if the existence of the former affects the behaviour, the value or the existence of the latter.

$$X \mathfrak{R} Y \Leftrightarrow X \triangleright Y, \text{ where } \triangleright \text{ signified that } X \text{ acts on } Y.$$

Notice that if *X maps Y*, *Y maps X* and vice versa. There is no reflexivity for the *represents* link.

These two links *maps* and *represents* allow to define correspondence between constructs. They are used in the definition of the fitness metrics.

## 2.3 Fitness Criteria and Metrics

In order to measure the degree of correspondence between components of the two ontologies we use fitness criteria and metrics. We adapted the Cavano and McCall framework [16] and organized the fitness measurement system in three levels, *factors*, *criteria* and *metrics*. As shown in Table 1 we identified four factors along which the fit can be measured namely, the *intentional factor*, the *informational factor*, the *functional factor*, and the *dynamic factor*. Each factor has associated *criteria*, which are in turn, related to *metrics* that allow the actual computation of the degree of fit. As highlighted in Table 1, the criteria and metrics are based on the *maps* and *represents* links between components of the SW and WW ontologies. Components are made explicit in the third and fourth column of the table. They are marked in italics in the short definition of the metrics whereas the type of link used in metrics is shown in bold.

Along the intentional dimension, the objective is to measure to which extent the system is meeting the business purpose. This is achieved by providing four associated to the *intentional factor* dealing, respectively, with the business activity and the goal purpose, and the actor and resource representation.

**Table 1.** Generic Fitness metrics

Factors	Criteria	SW construct	WW construct	Metrics
Intentional	Support ratio	Activity	Event	Number of <i>business activities represented</i> by system events / Number of <i>business activities</i>
	Goal satisfaction	Goal	States	Number of <i>goals for which each state constituting them maps a state</i> in the system / Number of <i>goals</i>
	Actor presence	Actor	Thing	Number of <i>business actors mapping</i> a system thing / Number of <i>business actors</i>
	Resource presence	Resource	Thing	Number of <i>business resources mapping</i> a system thing / Number of <i>business resources</i>
Informational	Information completeness	Thing	Thing	Number of <i>business things mapping</i> a system thing / Number of <i>business things</i>
	Information accuracy	States	States	Number of <i>business states mapping</i> a system state / Number of <i>business states</i>
Functional	Activity completeness	Thing	Thing	Number of <i>business things of a given activity mapping</i> a system thing / Number of <i>business things of this activity</i>
	Activity accuracy	States	States	Number of <i>business states of a given activity mapping</i> a system state / Number of <i>business states of this activity</i>
Dynamic	System reliability	Law	Law	Number of <i>business laws for which each business state maps a system state and the transformation between business states are possible between system states</i> / Number of <i>business laws</i>
	Dynamic realism	Path	States	Number of <i>paths for which each business state maps a system state and the succession of these system states is possible</i> / Number of <i>paths</i>

The *informational factor* complements the intentional factor by supporting a deeper analysis of the way activities are supported in the system. In order to provide a good fit between the system and the business processes, the system must (i) manipulate all the business process objects and (ii) support all the business process object states associated to the business processes. Two criteria have been defined in order to permit such an evaluation, the *Informational completeness* and the *Informational accuracy*, respectively.

The *functional factor* aims to measure the degree to which activities in the system correspond to business activities. The correspondence is based on involvement of things and their states in business and system activities, respectively. Each individual activity of a business process is so analysed separately using the *Activity completeness* and *Activity accuracy* based metrics.

The fourth factor, the *dynamic factor* aims to evaluate the extent to which the dynamicity of business processes is reflected in paths of system state transitions. It has two criteria namely, the *System reliability* and the *Dynamic realism* criterion.

All metrics are formally defined. As an example, let us present the *Support ratio* based metric:

$$\text{Support ratio (Sr)} = \frac{\text{Number of activities represented by system events}}{\text{Number of activities}}$$

A business activity is supported by the system if there exists an event in the system that *represents* it. Let:

- $A_b$  be the set of business activities (i.e. activities present in the business process),  $\text{card}(A_b)$  = the number of elements contained in  $A_b$ .
- $E_s$  be the set of system events

- $A_b^r$  be the set of business activities for which there exists a system event representing it;  $A_b^r = \{a, a \in A_b \mid \exists e \in E_s \wedge e \mathfrak{R} a\}$  and  $\text{card}(A_b^r)$  = number of elements contained of  $A_b^r$

Using these notations, the metric associated to this criterion is:

$$Sr = \text{card}(A_b^r) / \text{card}(A_b)$$

A complete definition of the generic fit measurement system can be seen in [17].

### 3 Specific Level: Generating a Specific Fit Measurement System

In this section, we show how the generic measurement system overviewed in the previous section guides the generation of a specific set of metrics. Specific metrics involve the correspondence between components of two specific models, to capture the business and the system, respectively. We selected the MAP representation formalism [14] for the former and the O\* model [15] for the latter. We present first, the generation process and secondly illustrate it in the case of MAP and O\*.

#### 3.1 The Generation Process

The generic fit measurement system presented in section 2 has two key components: (a) the two SW and WW ontologies, which identify components of interest in the representation of the business in one hand, and the system which supports it, on the other hand; (b) the set of criteria, which identifies correspondences between components of the WW and SW ontologies that are relevant to measure the fitness and metrics which compute the degree of correspondence and thus, the degree of fit.

The use of ontologies in this fitness measurement system is a way of being independent of the business and system models, thereby leading to a generic expression of the component correspondences and their associated metrics. Obviously, the generic system can serve as a mould to define a specific fitness measurement system thus, avoiding to redefine the relevant component correspondences and metrics for each specific set of models. This requires to establish the liaison between the set of specific models and the ontology set and then, derive the specific formulation of metrics.

This leads to the following three-steps generation process:

- Relate constructs of the chosen business model to those of the SW ontology.
- Relate constructs of the chosen system model and those of the WW ontology.
- Adapt the generic metrics.

It shall be noticed that step 1 and 2 concentrate on finding the concepts of a specific model which correspond to the ontology constructs involved in one (or several) fitness criteria. In other words, the instantiation of the ontology for a specific model can be limited to those parts which are relevant to perform step 3 of the generation process. Given the correspondence between an ontology construct and the

specific model concept, the metric formulae can be adapted easily. We illustrate this generation process in the following.

### 3.2 Generating the MAP/O\* Fitness Measurement System

**Relating MAP to SW.** The MAP representation system allows to represent a process model expressed in intentional terms. Goals (intentions) to be accomplished are explicitly represented in the process model together with the different alternative ways (strategies) for achieving them. MAP provides a representation mechanism based on a non-deterministic ordering of *intentions* and *strategies*. As shown in Fig. 6, a map is a labelled directed graph with intentions as nodes and strategies as edges between intentions. The directed nature of the graph shows which intentions can follow which one. An edge enters a node if its strategy can be used to achieve the intention of the node.

The key concepts of MAP are *intentions* (goals to achieve or maintain), *strategies* (means or manners to attain a goal) and *sections* which are triplets  $\langle I_i, I_j, S_{ij} \rangle$  where  $I_i$  is the source intention,  $I_j$  the target intention and  $S_{ij}$  the strategy to attain when  $I_i$  has been achieved. MAP includes a refinement mechanism by which a section in a map can be modelled as a map in its own. This leads to the representation of a business as a hierarchy of maps.

Formally, an intention  $I$  is defined as a set of *desirable states*  $G_I$  and every section has an *initial condition* and a *final condition*, both expressed in terms of states. A section  $S$  from intention  $I$  to intention  $J$  starts in a subset of states  $I_S \subseteq G_I$  and ends on a subset of states  $F_S \subseteq G_J$ .

MAP and SW share a goal-oriented view of a business process. However the latter does not employ a goal construct as an integral part of the model but as an external property whereas in the former goals are integral parts of the process model.

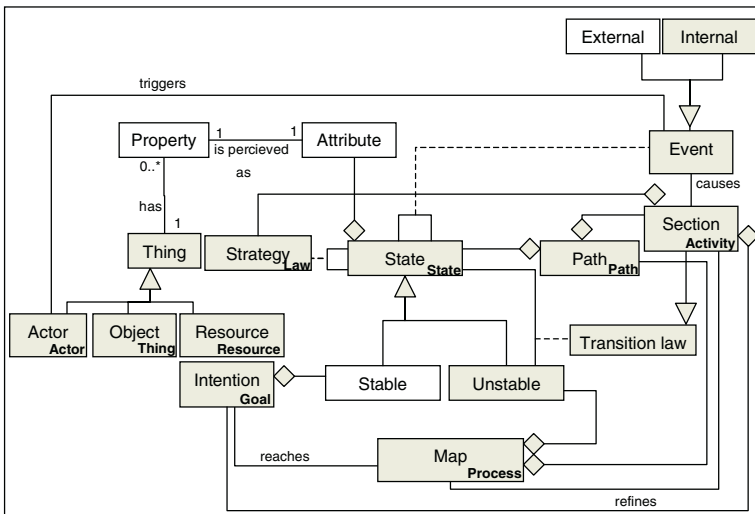


Fig. 3. Relating MAP to SW

Fig. 3 shows the instantiation of the MAP representation system in SW terms and highlights the correspondence between SW constructs and MAP concepts. To facilitate the reading of this correspondence in Fig. 3, we noted the names of the most important SW constructs (in particular, those involved in metrics) in the right corner of the boxes that represent the equivalent MAP concept in the figure. It can be seen that the *map* component in MAP corresponds to the *process* component in SW and that the *section* concept of MAP relates to the SW notion of *activity*. In both models the process is viewed as a *path* (of sections from Start to Stop in MAP and activities in SW). The MAP *strategy* involved in a section ensures a mapping between subsets of states, hence it specifies the SW *law*. A process model expressed as a map contains discontinuity related to *events* that can be *external* (a map is triggered by an *external event* arising in a thing and resulting from the action of some other thing called *actor*) or *internal* (section triggering). *Resources* (i.e. some things which do not take further actions) and *objects* appear in the map specification as parameters of the intention and strategy linguistic formulation.

**Relating O\* to WW.** O\* is a model which allows a conceptual representation of the system to be developed in an object oriented manner. In O\* an *object* is viewed as undertaking changes due to *events*. The specification of an object class therefore, goes beyond the traditional description with attributes and methods to include the description of events as state changes of objects which trigger *operations* on other objects, change their *states* and then, generate other events. O\* is based on a causal behavioural paradigm: events trigger operations which change states of object and generate state changes that maybe events which in turn trigger operations etc. Fig. 4 (a) illustrates this in a graphical mode. Fig. 4 (b) sketches the specification format of an O\* Class.

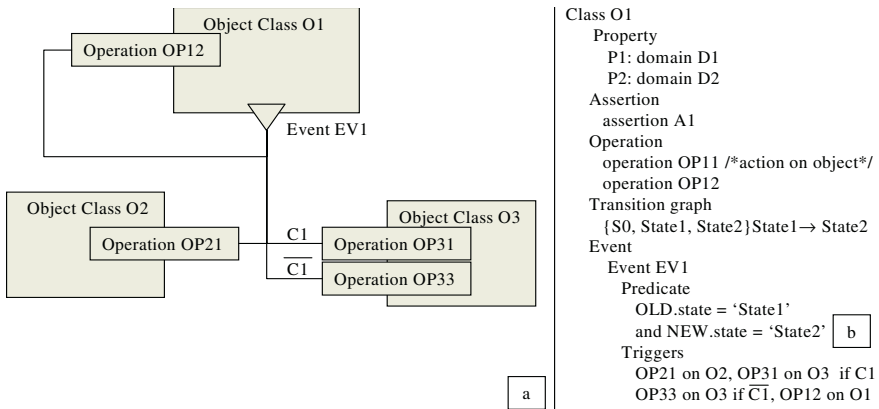


Fig. 4. Description of an object class

The WW ontology and the O\* model both allow to describe a system with a static and a dynamic point of view. The static part provides description of durable links between



constructs (such as composition), where as the dynamic part focuses on behavioural interactions between constructs or with the environment in response to events.

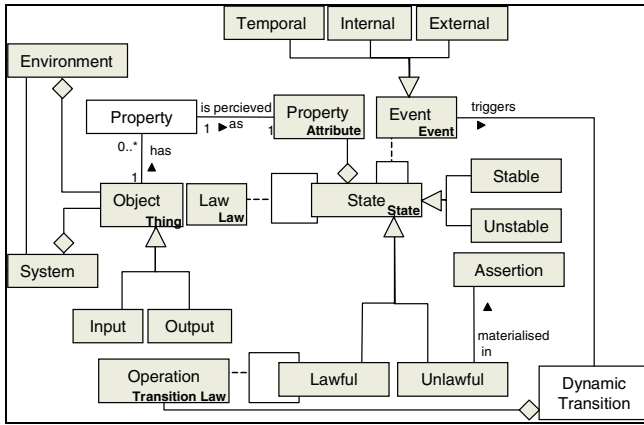


Fig. 5. Instantiation of the WW ontology with the O\* model

Fig. 5 presents the instantiation of the O\* model in WW terms and highlights the relationship between WW constructs and O\* concepts. As for the MAP instantiation, the names of WW constructs and particularly those involved in metrics are noted in the right corner of the boxes that represent the equivalent O\* concepts.

The *object* component in O\* relates to the WW *thing* construct. They both have *properties* and see their *states* change in response to *events*. The *event* in O\* can be *external* (an external event has for origin a stimulus coming from the environment of the system. It is associated to an actor), *internal* (it responds to a particular state change of a system object) or *temporal* (if it occurs at a foreseen point of time). The O\* notion of *operation*, which changes states, correspond to the *transition law* construct in the WW ontology. An O\* *assertion* corresponds to the WW construct of *unlawful state*.

**MAP/O\* Fit Measurement System.** Based on these relationships between the generic constructs of SW and WW ontologies and the MAP and O\* concepts the conversion of generic metrics into specific ones is easy to carry out.

Table 2 informally presents the ten metrics to measure the fit between a business expressed in MAP terms and the system specification expressed with the O\* language.

Let us considered the specific *Support ratio* metric in comparison to the generic one given in section 2.

In generic terms, the support ratio metric measures the degree of correspondence between the number of activities represented by system events and the total number of activities in the business process. At the specific level, the measure is established between the number of *map sections represented* by *events* and the total number of *map sections*. The generic formula provided in section 2 is adapted as follows:

**Table 2.** Specific metrics for measuring business/ system fit modelled in MAP and O\* terms

Criteria	MAP constructs	UML constructs	Specific metrics
Support Ratio	Section	Event	number of sections represented by events / number of sections
Goal Satisfaction	Intention	State	Number of intentions for which each state maps a state in the system / Number of intentions
Actor Presence	Actor	Object	Number of business actors mapping a system class / Number of business actors
Resource Presence	Resource	Object	Number of business resources mapping a system class / Number of business resources
Information Completeness	Object	Object	Number of business objects mapping system class / Number of business objects
Information Accuracy	State	State	Number of business states mapping to system states / Number of business states
Activity Completeness	Object	Object	Same as Information Completeness but for one given section
Activity Accuracy	State	State	Same as Information Accuracy but for one given section
System Reliability	Law	State	Number of business laws for which each business state maps a system state and the transformation between business states are possible between system states / Number of business laws
Dynamic Realism	Path	State	Number of paths for which each business state maps a system state and the succession of these system states is possible / Number of possible paths

Let:

- $S_b$  be the set of business sections,  $\text{card}(S_b)$  = the number of elements contained in  $S_b$ .
- $E_s$  be the set of events
- $S_b^r$  be the set of business sections for which it exists event representing them;  $S_b^r = \{ s, s \in S_b \mid \exists e \in E_s \wedge e \mathfrak{R} s \}$  and  $\text{card}(S_b^r)$  = number of elements contained in  $S_b^r$

Using these notations, the metric associated to the Support Ratio is:

$$Sr = \text{card}(S_b^r) / \text{card}(S_b)$$

All the metrics have been adapted in a similar manner. A brief summary is as follows.

The generic *Goal satisfaction* metric compares the number of goals supported by the system to the number of business goals. The goal as defined in the SW ontology corresponds to the MAP intention. An intention I is supported by the system if each state constituting the goal set  $G_I$  maps to a state of an object in the O\* model.

At the generic level, the *Actor presence* metric calculates the ratio of business actors present in the system on the total number of business actors. The construct of actor exists in the MAP model and is present in the system if it maps a system thing that triggers actions on another thing.

The generic *Information completeness* allows to measure to which extent a business thing maps a system thing. The SW and WW constructs of thing are related to the MAP object and O\* object respectively. A MAP object is supported by the system if there exists an O\* object that maps it.

The generic *Information accuracy* brings SW and WW states into play. These two constructs respectively correspond to MAP and O\* states. At the specific level, the Information accuracy metrics allows to compare the number of MAP states that map an O\* state to the total number of MAP states.

At the generic level, the *Activity completeness* and *Activity accuracy* provide information for a given activity on thing and states, respectively. The SW activity

construct corresponds to the MAP section. Thus, at the specific level, these criteria allow the analysis of a given section by calculating (1) the number of objects in the MAP section that *maps* to an O\* object by comparison with the total number of objects in the section and (2) the ratio of states occurring in the section that individually *maps* an O\* state.

The generic *System reliability* metric compares the number of business laws implemented in the system to the number of business laws. A business law is implemented in the system if each business state occurring in the law *maps* a system state and the transformations between these business states are possible between system states. The SW and WW constructs of state correspond to the MAP and O\* concept of state respectively. The System reliability metric is then identical to the generic metric but using MAP and O\* states.

The purpose of the generic *Dynamic realism* is to compare the number of paths present in the system to the number of paths. The path, as defined in the SW ontology corresponds to the MAP path. A path is present in the system if each state constituting it *maps* an O\* state and the succession of these system states is possible.

## 4 Applying Fitness Metrics in the Hotel Room Case Study

In this section, we illustrate the usage of the specific fitness measurement system in a hotel room booking case study.

### 4.1 Description of the Case Study

Competition with international hotel chains being always harder, the owners of several small hotels made the decision to become partners in order to offer attractive products and provide better services to their clients. They believe that offering packages of products combining room booking, sport activities and cultural manifestations will give them a competitive advantage. They consider important to facilitate the booking process as much as possible and to offer multiple different sale channels. Finally, they opted for both proactive and reactive strategies in order to attract new customers and to utmost satisfy the customers' needs.

These objectives are reflected in the business process modelled with MAP and presented in Fig. 6.

The map comprises two intentions: “*Offer packages*” and “*Manage customer relationship*” that are in line with the business decisions made. There are a number of strategies associated with each of these two intentions, particularly with the “*Manage customer relationship*”. These strategies reflect the desire to concentrate the business towards satisfying the customer, facilitating his/her booking and being proactive to ensure customer loyalty. The ‘*by offering booking facilities*’ strategy for example, is a cluster showing that booking can be done on the spot, by Internet or through an agency. ‘*By proactive offering*’ is a strategy which aims at selling a new booking to an old customer whereas the ‘*by rewarding*’ strategy contributes evidently to keep a customer loyal to the hotels group. The process terminates only if the client withdraws his booking or by necessity because his/her behaviour is reprehensible.

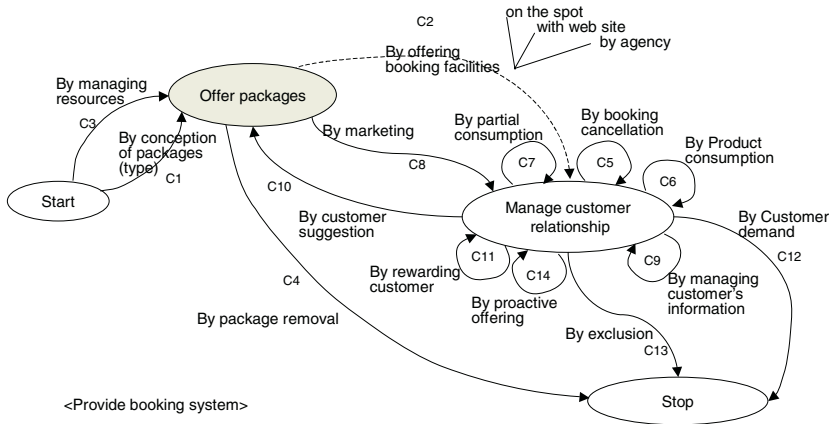


Fig. 6. The room booking business as a map

In order to get a complete understanding of business intentions and strategies, it was necessary to refine a number of sections of the above map. In total, we modelled five maps organised in two levels of abstraction. The complete specification includes 39 sections, 3 actors (the hotel keeper, the partner and the client), 6 objects (Hotel, Destination, Demand, Booking, Client, and Package) and 21 states.

#### 4.2 Measuring the Degree of Fitness

To develop the supporting information system, the hotel owners were having a restricted budget. They decided to adapt a legacy information system and to cope with their budget; they considered the three following options:

- Option 1: the system offers different sale channels to book a package and maintains the customer data over time. But the system does not manage pending requests. Thus, requests are dealt in real time and either transformed into bookings or abandoned.
- Option 2: the system handles pending requests but does not keep customer data. Clients must register at every booking.
- Option 3: combines the management of pending requests and customers data. However, the system does not manage automatically the reorganisation of a booking affected by events such as floods or typhoons that make the hotel resources temporarily unavailable.

In order to facilitate the choice of one of the three design options, we proposed to the hotel owners to measure the degree of fit in each of the three cases. We modelled the three options with O\* and then, applied the 10 metrics. Results are shown in Table 3 (note that the Activity completeness and Activity Accuracy criteria have been measured for the <Define, customer request, Management of the pending request, Through wait-listing> section).

**Table 3.** Fitness measures

Criteria	Design alternative 1	Design alternative 2	Design alternative 3
Support Rate	0,9(35/39)	0,9(35/39)	1 (39/39)
Goal Satisfaction	0,8 (4/5)	0,6 (3/5)	0,6 (3/5)
Actor Presence	1 (3/3)	1 (3/3)	1 (3/3)
Resource Presence	1 (2/2)	0,5 (1/2)	1 (2/2)
Information Completeness	1 (7/7)	0,86 (6/7)	0,86 (6/7)
Information Accuracy	0,91 (20/22)	0,86 (19/22)	0,86 (19/22)
Activity Completeness	1 (5/5)	0,8 (4/5)	1 (5/5)
Activity Accuracy	0,75 (3/4)	0,75 (3/4)	1 (4/4)
System Reliability	0,89 (54/61)	0,93 (57/61)	0,79 (48/61)
Dynamic Realism	0,79 (53/67)	0,53 (36/67)	0,84 (56/67)

### 4.3 Discussion

Table 3 considered at the glance, shows that none of the options provides a complete fit as a number of measures are inferior to 1. Option 3 is the only one showing a full system support of all business activities (support ratio = 1) but none of the three options completely supports the satisfaction of business goals (Goal satisfaction <1). The business actors representation is good in the three options (Actor presence =1) but the resource fit is low in option 2. Thus, measures demonstrate difference in the *Intentional* fit provided by the different options. Vice versa, measures shows that the three options deal in a similar manner with the *Informational factor* with an advantage to option 1 where all business objects are represented and managed by the system (Information completeness =1). Along the *Functional factor*, option 3 is the most fitting solution for the given section as the Activity completeness measure and the Activity accuracy equal 1. Finally, measures along the *Dynamic factor* are all inferior to 1. In all options some business states do not exist in the system (Goal satisfaction < 1) thus implying low measures of the fit related to dynamic aspects (System reliability and dynamic realism).

Let us complement this overall evaluation of options by a more in depth reasoning based on the fitness measures. Option 3 is appealing because the support ratio equals 1, i.e. the system supports every business activity. However, other measures have to be considered because each criterion brings a different viewpoint on the degree of fit. Two design options can have the same value for a fitness criterion (e.g. *Support ratio* equals 34/39 in option 1 and 2) and different ones for other criteria (five criteria have different values for these two options). Option 3 has against it the low measure of Goal satisfaction fit which in turn, implies a lower value of dynamic fitness measures compared to option 1.

The decision about the option to implement can be based on its ability to evolve in the future and the fitness metrics can help in this long term perspective. A better fit requires either a business adaptation or a system change. For example, it can happen that some planned strategy in the map of Fig. 8 reveals inefficient in practice and will be abandoned. Vice versa, a better Information completeness measure in option 3 can be achieved by adding the representation of one or more business objects. Thus, taking into account the cost of adding new system objects and its subsequent impact on the improvement of measures in the future can help making the decision now in favour of option 3.

There are dependencies between criteria and these have to be taken into account during this selection phase. For example, if some SW things do not *map* WW things, the states of these things cannot *map* WW states. Therefore, the *Information completeness* value influences the *Information accuracy* value. In the same way, the absence of *mapping* of states influences *Goal satisfaction*, *System reliability* and *Dynamic realism* measures. The improvement of the Goal satisfaction criterion in Option 3 will contribute to raise the value of the dynamic fit.

The introduction of thresholds for measures can facilitate decisions. Thresholds can be determined by stakeholders in order to avoid a situation of unacceptable unfit. In our case, assume that the threshold for the Dynamic realism criterion has been established at 0.8. Options 1 and 2 will be eliminated on this basis.

Finally, it is possible for stakeholders to allocate *weights* to the different activities involved in the business to reflect priority given to certain activities against others. Priority can be given for example, to client satisfaction, profitability, activity frequency, etc. Weighting activities related to customer satisfaction in our case showed that option 3 was the best over the three options and decision was made to implement it. Indeed, option 3 was having a competitive advantage as the business parts weakly represented in the system have no impact on clients. An analysis of the Activity completeness and Activity accuracy criteria strengthens this position. Furthermore, by introducing weighting depending on activity frequency confirmed the choice for option 3. Indeed, the measures help stakeholders realise that exceptional events which have to be handled manually in option 3 are relatively rare and that the low fit due to these can be accepted.

## 5 Conclusion

In this paper, a process for generating metrics to evaluate the fit between specific business and system models was presented and illustrated. This process uses a set of generic criteria and metrics as a mould for producing the specific fit measurement system. The criteria and metrics developed adopt fit measurement that takes the business view as a reference and compare the system view with it. Therefore in any metric, the denominator always refers to constructs of the system model. This process has been used to generate fitness metrics for the MAP and O\* models that respectively represent the business and the system.

We used the generated metrics to show how fitness measures during system design can feed back to improve the fit of the system-under-construction. We considered three design options in the construction of a system to support hotel room booking and based the choice of the option to be implemented on the fit measures. We showed that for each criterion, it is possible to define a threshold value: If the metric determines a value lower than the associated threshold, the business process and supporting software systems are misfitting, motivating corrective action. This action can be a modification of either the business model or the system. Thus thresholding leads to a beneficial cycle of design-measurement-design for better fit.

In order to better take into account the characteristics of each project in the calculation of the fitness measures, we intend to explore the weighting technique that

we illustrated in the case study. This allows to attribute relative importance to the different constructs of the SW ontology, according to, for example, the added value, the customers satisfaction, the frequency... For example, the definition of the *Support ratio* considers the number of automated activities and ignores the relative value addition of activities in the business. Thus, it can happen that the support ratio is high but the most value adding activities are not automated. Appropriate weighting obviates this problem.

Our research agenda relies on two key issues: (i) the use of the fitness measurement system in a context of evolution and (ii) the development of a tool to support the proposed process and the calculation of the fitness measures for a given project.

## References

- [1] Giaglis GM (2001) A Taxonomy of Business Process Modelling and Information Systems Modelling Techniques. *Journal of Flexible Manufacturing Systems* 13 (2), pp. 209-228.
- [2] Fifth Workshop on Business Process Modeling, Development, and Support BPMDS'04 (2004), Riga, Latvia.
- [3] Sabherwal, R. and Y. E. Chan (2001). Alignment Between Business and IS Strategies: A Study of Prospectors, Analyzers, and Defenders. *Information Systems Research*, March, 12(1): 11-33.
- [4] Henderson, J. C. and N. Venkatraman (1993). Strategic Alignment: Leveraging Information Technology for Transforming Organizations. *IBM Systems Journal*, 32(1): 4-16.
- [5] Munford, E. (1981) Participative Systems Design: Structure and Method systems, Objectives, Solution, Vo. 1, North-Holland, 5-19.
- [6] Rubin KS, Goldberg A (1992) Object Behavior Analysis, *Communications of the ACM*, vol 35, N°9.
- [7] Dardenne A., Lamsweerde A., Fickas, S. (1993): *Goal-directed Requirements Acquisition*, Science of Computer Programming, 20, Elsevier, pp.3-50.
- [8] Soffer P (2004) Fit Measurement: How to Distinguish Between Fit and Misfit, note for BPMDS'04, Riga, Latvia
- [9] Regev G, Wegmann A (2004) Remaining Fit: On the Creation and Maintenance of Fit. Proceedings of BPMDS'04, Riga, Latvia, 2004.
- [10] Wand Y, Weber R (1992) An Ontological Model of an Information System, *IEEE Transactions on Software Engineering*, November, pp. 1282-92
- [11] Soffer P, Wand Y (2004) Goal-Driven Analysis of Process Model Validity. Proceedings of CAiSE'04, Riga, Latvia.
- [12] Bunge, M (1977) *Treatise on Basic Philosophy: Ontology I. The Furniture of the World*, Reidel.
- [13] Bunge, M (1979) *Treatise on Basic Philosophy: Ontology II. A World of Systems*, Reidel.
- [14] Rolland C., Prakash N. (2001) Matching ERP System Functionality to Customer Requirements, Proceedings RE'01, Toronto, Canada, pp. 66-75.
- [15] Lee S.P., Rolland C., Brunet J.. Abstraction in an Object-Oriented Analysis Method. in *Malaysian Journal of Computer Science*, Vol. 10, No 1, June 1997, p. 53-63.

- [16] Cavano J.P., McCall J.A (1978) A Framework for the Measurement of Software Quality. Proceedings of the Software Quality and Assurance Workshop, San Diego, pp. 133–139,.
- [17] Etien A. and Rolland C. (2005) Measuring the Fitness relationship, submitted to the special issue Coordinated Development of Business Processes and their Support Systems of the Requirements Engineering Journal



# A Concern-Oriented Requirements Engineering Model

Ana Moreira<sup>†</sup>, João Araújo<sup>†</sup>, and Awais Rashid<sup>‡</sup>

<sup>†</sup> CITI/Dept. Informática, FCT, Universidade Nova de Lisboa,  
2829-516 Caparica, Portugal

<sup>‡</sup> Computing Department, Lancaster University,  
Lancaster LA1 4YR, UK

{amm, ja}@di.fct.unl.pt, awais@comp.lancs.ac.uk

**Abstract.** Traditional requirements engineering approaches suffer from the tyranny of the dominant decomposition, with functional requirements serving as the base decomposition and non-functional requirements cutting across them. In this paper, we propose a model that decomposes requirements in a uniform fashion regardless of their functional or non-functional nature. This makes it possible to project any particular set of requirements on a range of other requirements, hence supporting a multi-dimensional separation. The projections are achieved through composition rules employing informal, often concern-specific, actions and operators. The approach supports establishment of early trade-offs among crosscutting and overlapping requirements. This, in turn, facilitates negotiation and decision-making among stakeholders.

## 1 Introduction

The tyranny of the dominant decomposition [21] refers to the limited mechanisms used by traditional methods to decompose complex systems into separate concerns. Modern approaches propose mechanisms for decomposition and composition. However, they mostly use a dominant dimension as the base decomposition, with other possible dimensions cutting across them. For example, approaches, such as the NFR framework [2], use non-functional requirements as the dominant dimension with the functional dimension added a posteriori. Other existing requirements engineering (RE) approaches, such as viewpoints [7, 19] and use cases [9], use functional requirements as the dominant decomposition with analysis conducted against a set of non-functional requirements cutting across the base.

It has been argued that crosscutting is a phenomenon that is not limited to non-functional requirements and that functional requirements can also often cut across parts of a system [16]. Existing separation of concerns mechanisms at the RE level do not explicitly account for such crosscutting nature of functional requirements. Consequently, they cannot be handled effectively leading to a lack of identification and characterisation of their influence on other concerns in the system. Furthermore, an initially non-crosscutting set of requirements (functional or non-functional) might become crosscutting in future. The two-dimensional nature of existing decomposition approaches does not provide support to deal with such unanticipated evolution.

In this paper, we propose a model that decomposes requirements in a uniform fashion regardless of their functional or non-functional nature. This makes it possible to *project* any particular set of requirements on a range of other requirements, hence supporting a multi-dimensional separation. A projection specifies the influence of a given concern on other concerns and is achieved through composition rules employing informal, often concern-specific, actions and operators. The rules specify the projection of a particular concern onto other concerns it relates to. The various projections make it possible for us to compose a range of *reflected projections* contributing to an individual concern. The approach supports establishment of early trade-offs among crosscutting and overlapping requirements. This, in turn, facilitates negotiation and decision-making among stakeholders. The uniform nature of the decomposition also makes it possible to deal with situations where an initially non-crosscutting set of requirements evolves to have a wider influence in the system.

Section 2 discusses existing approaches to separate crosscutting concerns at the RE level and highlights how these suffer from the tyranny of dominant decomposition. Section 3 presents our model for multi-dimensional separation of requirements level concerns. Section 4 provides an overview of the realisation of the model using XML and applies it to a case study of a location and context sensitive tourist guide. Section 5 discusses some related work, while Section 6 concludes the paper and identifies directions for future work.

## 2 Background

Separation of concerns has been contemplated by well-known RE approaches such as goal-oriented techniques and viewpoints. In goal-oriented approaches [12], such as KAOS [3] and *i\** [23], a goal is an objective that the system under consideration should achieve. It can be formulated at different levels of abstraction and covers concerns in two dimensions, i.e., functional and non-functional. KAOS uses a formal language (first-order temporal logic with real-time constraints) to specify critical parts of the system, besides allowing informal modelling. Goals are used to detect and manage conflicts among requirements. The *i\** framework identifies and models organisational requirements and adopts the goal and softgoal modelling concepts as its dimensions. A softgoal represents a non-functional requirement we expect to satisfy within acceptable limits.

Separation of crosscutting properties has also been considered in PREView [19], a viewpoint-oriented requirements engineering method. A PREView viewpoint encapsulates partial information about the system. Requirements are organised in terms of several viewpoints, and analysis is conducted against a set of concerns intended to correspond broadly to the overall system goals. In applications of the method, the concerns that are identified are typically high-level non-functional requirements. Here again the separation of concerns is two-dimensional: one being the viewpoints that handle functional requirements and the PREView-specific notion of concerns which encapsulate non-functional properties.

The Aspect-Oriented Requirements Engineering (AORE) model presented in [17] is based on treating PREView concerns as adaptations of the aspect-oriented programming [6] notion of aspects and, consequently, carries out the analysis of

broadly scoped properties against a base set of viewpoints. A refinement of this model, presented in [16], supports separation of the specification of aspectual requirements, non-aspectual requirements and composition rules in modules representing coherent abstractions and following well-defined templates. This modularisation makes it possible to establish early trade-offs between aspectual requirements hence providing support for negotiation and subsequent decision-making among stakeholders. However, the composition rules have to be written with reference to a dominant decomposition that aspects cut across.

The discussion above demonstrates that, while existing RE approaches support analyses of system requirements from the perspective of non-functional properties, support for identifying the influence of crosscutting functional properties (or a combination of functional and non-functional properties) is not available. Nor is there any support for incorporating such an influence during trade-off analysis and subsequent negotiation among stakeholders. The multi-dimensional approach presented in this paper addresses the above issues by eliminating the dominant decomposition through uniform treatment of the various types of requirements in the system. In deriving our multi-dimensional model we have built on the strengths of the model in [16], mainly the informal composition rules with concern-specific actions and operators and the effective support for establishing trade-offs and negotiations among stakeholders.

### 3 A Concern-Oriented Model for RE

Modern systems have to run in highly volatile environments where the business rules change rapidly. Therefore, systems must be easy to adapt and evolve. In order to facilitate adaptability and evolution, it is essential that the influence of any set of requirements on the system can be determined. In existing RE approaches, such analyses focus on the influence of non-functional requirements. Functional requirements that might have a wide impact on other functional or non-functional requirements are not effectively dealt with. In Section 2, we argued that this is a direct consequence of having a largely two-dimensional decomposition.

Our proposed model addresses this problem by treating all concerns in a uniform fashion. Concerns in our model imply any coherent collection of requirements. We do not classify concerns into viewpoints, use cases or aspects though our concerns still encapsulate coherent sets of functional and non-functional requirements. As shown in Figure 1, we perceive the concern space at the requirements level as a hypercube. Each face of the hypercube represents a particular concern of interest. By treating all concerns as equal we can choose any set of concerns as a base to project the influence of another concern or set of concerns onto this base. This flexible, multi-dimensional view makes it possible to handle both crosscutting functional and non-functional requirements in an effective fashion.

Our RE model is shown in Figure 2. We start by identifying and specifying concerns. Concern identification is carried out using a synthesis of existing requirements elicitation mechanisms such as viewpoints [7], use cases [9] and goals [12]. The identified concerns are specified using well-defined templates (cf. Section 4.1.1).

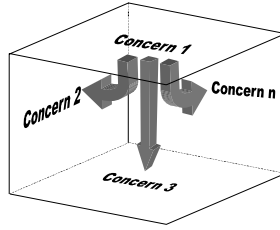


Fig. 1. Concern space represented as a hypercube (the block arrows represent projections)

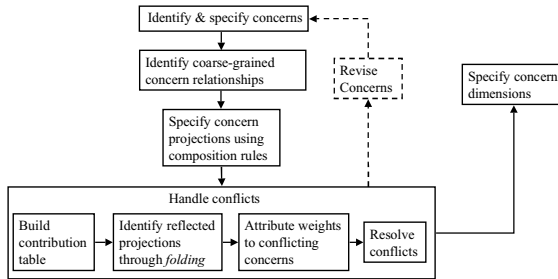


Fig. 2. RE model based on uniform treatment of concerns

The next step is to identify coarse-grained relationships among concerns by relating concerns to each other through a matrix. These relationships are identified using techniques such as domain analysis [10], ethnography [22] and natural language processing [18]. Looking at the matrix (cf. Table 1) we can see which concerns influence other concerns and whether any reciprocal influence exists.

Notice that in this paper, we do not focus on the exact kind of relationships between two concerns. In [3] interested readers can find a model for requirements interdependencies and inter-relationships.

Once the coarse-grained relationships between concerns have been established, the next step is to specify the possible projections of each concern on other concerns. This is achieved through composition rules. These rules operate at the granularity of individual requirements and not just the concerns encapsulating them. Consequently, it is possible to specify how a requirement in the concern in question influences or constrains the behaviour of a set of requirements in various other concerns. At the same time, if desired, trade-offs among concerns can be observed at a finer granularity. This alleviates the need for unnecessary negotiations among stakeholders for cases where there might be an apparent trade-off between two (or more) concerns but, in fact, different, isolated requirements are being influenced by them. It also facilitates identification of individual, conflicting requirements with respect to which negotiations must be carried out and trade-offs established.

After specifying the various projections with the aid of composition rules, identification and resolution of conflicts among the concerns is carried out. This is accomplished by:

1. Building a contribution matrix (cf. Table 2) where each concern may contribute negatively (-) or positively (+) to the others (empty cells represent “don’t care” contributions). The diagonal is marked with the concern names to support observation of reflected projections in step 2. This matrix is inspired on the NFR framework [2].
2. *Folding* the table along its diagonal (cf. Figure 3) to obtain the cumulative effect for situations where two concerns directly influence each other. An example of this folding is shown for  $C_1$  and  $C_{n-1}$  in Figure 3. The folded table provides us the reflected projections: the combined influence of a set of concerns on a particular concern.
3. Attributing weights to those concerns that contribute negatively to each other in relation to a particular concern. Each weight is a real number in the interval  $[0 .. 1]$  and represents the priority of a concern in relation to the concern it is projected on.
4. Solving the conflicts with the stakeholders, using the above prioritisation approach to help negotiation and decision-making.

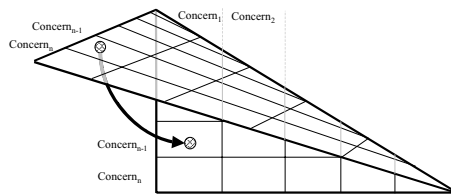
**Table 1.** Relating concerns to each other

	$C_1$	$C_2$	...	$C_n$
$C_1$		√		
$C_2$				√
...				...
$C_n$		√		

**Table 2.** Contributions between concerns

	$C_1$	$C_2$	...	$C_n$
$C_1$		+		
$C_2$				-
...				
$C_n$		-		

Conflict resolution might lead to a revision of the requirements specification (concerns and/or composition rules). If this happens, then the projections are revised and any further conflicts arising are resolved. The cycle is repeated until all conflicts have been resolved through effective negotiations.



**Fig. 3.** The concern contribution table folded along its diagonal

The last activity in the model is identification of the dimensions of a concern. As observed in [17], concerns at this early stage can have an impact on artefacts at later development stages that can be described in terms of two dimensions:

- *Mapping*: a concern might map onto a system feature/function (e.g., a simple method, object or component), decision (e.g., a decision for architecture choice) and design (and hence implementation) aspect (e.g., mobility cf. Section 4.1.1).

Note that despite their crosscutting nature at this stage, some concerns might not directly map onto an aspect at later stages.

- *Influence*: a concern might influence different points in a development cycle, e.g., availability influences the system architecture while mobility influences specification, architecture, design and implementation.

## 4 Realisation of the Model

We have employed the eXtensible Markup Language, XML, as the definition language to specify the concerns and the composition rules to relate them with each other. The concerns and composition rules are specified using pre-defined templates. These templates can, optionally, be enforced using XML schemas. XML has been chosen because, as demonstrated by the following case study, there is a need for concern-specific actions and composition operators when defining the composition rules. The extensible model offered by XML coupled with the rich specification model of the XML schema language makes it an ideal choice as it is virtually impossible to anticipate the various types of composition operators and actions that might be required. Since the XML schema language is extensible – it is based on XML itself – it is possible to enforce constraints on the specification of composition rules when new operators and/or actions are introduced. Furthermore, the ability to define semantically meaningful tags and informal operators ensures that the readability of the requirements specification is not compromised as the specification resides in the stakeholders' domain and must be readable by them.

The use of XML makes it possible to select any projections of interest by using XPath queries and observe their cumulative effect. The selected projections and their effect can also be visualised using the eXtensible Stylesheet Language (XSL). This aids scalability in the presence of a large number of concerns.

### 4.1 Case Study

The case study we have chosen is a location and context sensitive tourist guide system inspired by a real system implemented at Lancaster [5]:

“The system provides an electronic hand-held guide that offers the following facilities to the visitors: (1) retrieve information about the city, including information about their current location; (2) provide route guidance to help visitors move between locations on the tour; (3) enter a set of preferences and interests to generate suitable tours of the city; (4) access external services, such as hotel and theatre ticket reservations.”

#### 4.1.1 Identify and Specify Concerns

There are some concerns that are probably easier to identify as they directly represent stakeholders views on the basic functionality of the system. For example, we definitely have a concern that reflects the visitor needs and another for tourist information centre. Other concerns reflect more global properties of the system. For example, mobility will be a concern, as the system needs to react while the visitor is moving around. This brings immediately to our minds the context concern as the system needs to recognise the visitor's change in location. This, in turn, suggests portability as another one, since the visitor needs to carry with her/himself the

electronic device to access the system while on the move. Other concerns such as compatibility and availability are two obvious ones, since the system must be compatible with other external services (hotel and ticket reservation systems) and available anytime the visitor is using it. The concerns we identified are as follows:

- *Visitor*: users that can retrieve information from the system, including their current location.
- *Tourist Information Centre*: decides which information goes into the system.
- *Electronic Device*: used by the visitors to access the system.
- *Portability*: the electronic devices to access the system must be carried around by the visitors and therefore must be portable.
- *Mobility*: the system must handle mobility as the visitor will need to access the system on the move during his/her tour.
- *Context*: the system must recognise and handle the visitor's change in location.
- *Compatibility*: the system must be compatible with the external services it has to interact with, in particular, hotel and theatre ticket reservations.
- *Availability*: the system must always be available to react to stimuli (e.g., be accessed by the visitor) and for data updates.

We will be using the concerns *Visitor*, *Mobility* and *Compatibility* to illustrate our approach. Figures 4 through 6 show these concerns specified in XML.

```

<?xml version="1.0" ?>
- <Concern name="Visitor">
  - <Requirement id="1">
    The visitor will be able to retrieve information from the system.
    <Requirement id="1.1">
      The visitor will be able to access information about the attractions.
    </Requirement>
    - <Requirement id="1.2">
      The visitor will be able to access information about his/her location.
      <Requirement id="1.2.1">
        The visitor will be able to validate the information about the location if it does not correspond to what s/he sees.
      </Requirement>
    </Requirement>
    - <Requirement id="1.3">
      The visitor will be able to obtain a list of available preset tours.
    </Requirement>
  </Requirement>
- <Requirement id="2">
  The visitor will be able to create a custom tour.
  <Requirement id="2.1">
    The visitor will be able to specify preferences.
  </Requirement>
</Requirement>
- <Requirement id="3">
  The visitor will be able to follow a tour.
  <Requirement id="3.1">
    The visitor will be able to reconfigure the tour.
  </Requirement>
</Requirement>

```

```

- <Requirement id="4">
  The visitor will be able to access external services.
  <Requirement id="4.1">
    The visitor will be able to access hotel reservation.
  </Requirement>
  <Requirement id="4.2">
    The visitor will be able to access theatre ticket reservation.
  </Requirement>
</Requirement>
</Concern>

```

Fig. 4. The *Visitor* concern in XML

```

<?xml version="1.0" ?>
- <Concern name="Mobility">
  - <Requirement id="1">
    The system will be accessed on the move.
    <Requirement id="1.1">
      The system will be accessed from within a limited area.
    </Requirement>
  </Requirement>
</Concern>

```

Fig. 5. The *Mobility* concern in XML

```

<?xml version="1.0" ?>
- <Concern name="Compatibility">
  - <Requirement id="1">
    The system must be able to interact with external services.
  </Requirement>
</Concern>

```

Fig. 6. The *Compatibility* concern in XML

The structure is self-explanatory: a Concern tag denotes the start of a concern while a Requirement tag denotes the start of a requirement. Refinements such as sub-requirements are represented via the nesting of the tags. Each requirement has an id which is unique within its defining scope i.e. the concern. Concern names are unique within the case study. However, XML namespaces can be used for the purpose as well.

### 4.1.2 Identify Coarse-Grained Concern Relationships

As we identify and describe concerns we can relate them, by building the matrix in Table 3. The tick indicates a unidirectional relationship, from left to right, between two concerns. For example, *Tourist Information Centre* has an impact on *Visitor*, as it is responsible to make available the information visitors can access. Between *Visitor* and *Mobility*, on the other hand, we can identify two unidirectional relationships (as they are semantically different cf. composition rules in Section 4.1.3): one from *Visitor* to *Mobility* indicating that visitors require mobility; and another from *Mobility* to *Visitor* indicating that mobility has to support information access for visitor.

### 4.1.3 Specify Concern Projections Using Composition Rules

Having studied the impact of each concern on all the others we can now start by analysing in more detail each relationship. The fundamental idea is that we can *project* each concern on all the others with which the first has a relationship. The projection specifies the influence of a given concern (represented in a row in Table 3) on other concerns (represented in columns in Table 3). Whenever a concern affects several other concerns it has a broadly scoped impact on the system and, therefore, can be classified as a crosscutting concern. As we can see from Table 3, not only non-functional concerns such as *Mobility* are crosscutting but also functional concerns such as *Visitor* have a similar nature.

**Table 3.** Matrix relating concerns (**Vis:** Visitor; **TIC:** Tourist information centre; **Port:** Portability; **Mob:** Mobility; **ED:** Electronic device; **Cont:** Context; **Comp:** Compatibility; **Avail:** Availability)

Concerns Concerns	Vis	TIC	Port	Mob	ED	Cont	Comp	Avail
Vis	✓			✓	✓			
TIC	✓							
Port			✓	✓	✓			
Mob	✓		✓			✓		✓
ED								
Cont								
Comp	✓	✓		✓				
Avail	✓	✓		✓	✓			

The materialisation of these projections is accomplished here by defining a set of composition rules, one for each projection. Composition rules define the relationships between concerns requirements at a fine granularity (unlike the relationship matrix in Section 4.1.2 which is aimed at identifying coarse-grained relationships). Composition rule definitions can be governed by an XML schema. However, for simplification we describe the structure of composition rules with reference to some examples and not the XML schema definition. As shown in Figures 7 through 9, a



coherent set of composition rules is encapsulated in a Composition tag. Figure 7 encapsulates all compositions (i.e. projections) for the *Visitor* requirements while Figures 8 and 9 do so for *Mobility* requirements and *Compatibility* requirements respectively. The semantics of the Requirement tag here differ from the tags in the concern definition. If a concern requirement has any sub-requirements these must be explicitly excluded or included in the Constraint imposed by a concern requirement. This is done by providing an “include” or “exclude” value to the optional children attribute. A value of “all” for a concern or id value implies that all the requirements within the specified concern are to be constrained.

The Constraint tag defines an, often concern-specific, action and operator defining how the concern requirements are to be constrained by another concern requirement. Although the actions and operators are informal, they have clearly defined meaning and semantics to ensure valid composition of concerns. This provides the architects and designers a systematic means to interpret the requirements specification. The Outcome tag defines the result of constraining the concern requirements with another concern requirement. The action value describes whether another concern requirement or a set of concern requirements must be satisfied or merely the constraint specified has to be fulfilled (see Table 6).

```

<?xml version="1.0" ?>
<Composition>
  <Requirement concern="Visitor" id="all">
    <Constraint action="ensure" operator="during">
      <Requirement concern="Mobility" id="1" children="include" />
    </Constraint>
    <Outcome action="fulfilled" />
  </Requirement>
  <Requirement concern="Visitor" id="all">
    <Constraint action="provide" operator="by">
      <Requirement concern="ElectronicDevice" id="all" />
    </Constraint>
    <Outcome action="fulfilled" />
  </Requirement>
</Composition>

```

Fig. 7. Composition rule for *Visitor*

```

<?xml version="1.0" ?>
<Composition>
  <Requirement concern="Mobility" id="1" children="include">
    <Constraint action="provide" operator="for">
      <Requirement concern="Visitor" id="all" />
    </Constraint>
    <Outcome action="fulfilled" />
  </Requirement>
  <Requirement concern="Mobility" id="1" children="exclude">
    <Constraint action="enforce" operator="for">
      <Requirement concern="Portability" id="1" />
    </Constraint>
    <Outcome action="fulfilled" />
  </Requirement>
  <Requirement concern="Mobility" id="1" children="include">

```

```

    <Constraint action="affect" operator="on">
      <Requirement concern="Context" id="1" children="include" />
    </Constraint>
    <Outcome action="fulfilled" />
  </Requirement>
  <Requirement concern="Mobility" id="1.1">
    <Constraint action="affect" operator="on">
      <Requirement concern="Availability" id="all">
    </Constraint>
    <Outcome action="fulfilled" />
  </Requirement>
</Composition>

```

Fig. 8. Composition rule for *Mobility*

```

<?xml version="1.0" ?>
<Composition>
  <Requirement concern="Compatibility" id="1">
    <Constraint action="ensure" operator="with">
      <Requirement concern="Visitor" id="4" children="include" />
    </Constraint>
    <Outcome action="satisfied">
      <Requirement concern="Mobility" id="1" children="include" />
    </Outcome>
  </Requirement>
  <Requirement concern="Compatibility" id="1">
    <Constraint action="ensure" operator="with">
      <Requirement concern="TouristInformationCentre" id="2" />
    </Constraint>
    <Outcome action="fulfilled" />
  </Requirement>
</Composition>

```

Fig. 9. Composition rule for *Compatibility*

The informality of the actions and operators ensures that the composition specification is still readable by the stakeholders, an important consideration during

requirements engineering. For example, if we look at the first composition rule in Figure 7 and focus on the values in bold we get the following: “**All Visitor** requirements must be **ensured during** requirement **1** of **Mobility**, **including** its children, with the outcome that the Visitor’s requirements are **fulfilled**”.

Tables 4 to 6 describe the semantics of the actions and operators, which we have defined so far, for Constraint and Outcome. The initial set of these actions and operators was first defined in [16]. Here, we have validated that proposal and identified one new action: *affect*.

**Table 4.** Description of *Constraint* actions

Constraint Action	
Type	Description
enforce	Used to impose an additional condition over a set of concern requirements.
ensure	Used to assert that a condition that should exist for a set of concern requirements actually exists.
provide	Used to specify additional features to be incorporated for a set of concern requirements.
applied	Used to describe rules that apply to a set of concern requirements and might alter their outcome.
exclude	Used to exclude some concerns or requirements if the value <i>all</i> is specified.
affect	Used to specify that a set of concern requirements will alter the state of another concern.

**Table 5.** Description of *Constraint* operators

Constraint Operator	
Type	Description
during	Describes the temporal interval during which a set of requirements is being satisfied.
between	Describes the temporal interval falling between the satisfaction of two requirements. The interval starts when the first requirement is satisfied and ends when the second one is to start being satisfied.
on	Describes the temporal point after a set of requirements has been satisfied.
for	Describes that additional features will complement the concern requirements.
with	Describes that a condition will hold for two sets of requirements with respect to each other.
in	Describes that a condition will hold for a set of requirements that has been satisfied.
AND,OR, XOR	Conjunction, disjunction and exclusive-OR (when either requirement is satisfied but not both)

The interesting point to note here is that not all operators are concern-specific, e.g. XOR is a generic operator. Also, the actions for the Outcome are generic and not specific to a particular concern. It is, however, not possible to say whether Outcome actions are always generic, as more case studies need to be carried out before arriving at such a conclusion. It is also worth noting that although the same operator might apply to different concern requirements, not all operator-action combinations are valid in the Constraint specification for a particular concern. More case studies need to be carried out to validate the set of operator-action combinations.

**Table 6.** Description of *Outcome* actions

Outcome Action	
Type	Description
satisfied	Used to assert that a set of viewpoint requirements will be satisfied after the constraints of a concern requirement have been applied.
fulfilled	Used to assert that the constraints of a concern requirement have been successfully imposed.

**4.1.4 Handle Conflicts**

The composition rules leads to the identification of conflicts among concerns whose requirements constrain the same or overlapping sets of other concern requirements. In case of our approach this process is optimised as any potential interaction or conflict can be deduced from the composition rules. Conflict resolution is carried out in the four steps described below.

**Build the Contribution Table.** The contribution table (cf. Table 7) shows in which way (negatively or positively) a concern contributes to the others. Each cell shows a unidirectional contribution between a concern located in a line and a concern located in a column. In this case, *Availability* contributes positively to *Visitor* and *Tourist Information Centre* and negatively to *Mobility* and *Electronic Device*.

**Table 7.** Contribution table

Concerns Concerns	Vis	TIC	Port	Mob	ED	Cont	Comp	Avail
Vis				+	+			
TIC	+							
Port				+	+			
Mob	+		+			+		-
ED								
Cont								
Comp	+	+		-				
Avail	+	+		-	-			

**Identify Reflected Projections Through Folding.** Having studied the contributions between concerns we can now fold the table in order to reduce the range of projections we have to deal with. During folding, concerns that have a symmetric projection on each other have their effects accumulated. This is shown in Table 8. The columns in the table show the reflected projections of various concerns on an individual concern.

Table 8 shows that *Compatibility* and *Availability* contribute negatively to *Mobility*. We can help resolve such conflict by attributing weights to the concerns involved in the conflicting situation.

**Table 8.** Folded table w/ reflected projections

Vis	Vis							
TIC	+	TIC						
Port			Port					
Mob	+		+	Mob				
ED	+		+		ED			
Cont				+		Cont		
Comp	+	+		-			Comp	
Avail	+	+		-	-			Avail

**Attribute Weights to Conflicting Concerns.** Weighting allows us to describe the extent to which a concern may constrain another. The values are given according to the importance each concern has with respect to another one. The scales we are using are based on ideas from fuzzy logic and have the following meaning:

- *Very important* takes values in the interval ] 0,8 .. 1,0]
- *Important* takes values in the interval ] 0,5 .. 0,8]
- *Average* takes values in the interval ] 0,3 .. 0,5]
- *Not so important* takes values in the interval ] 0,1 .. 0,3]
- *Do not care much* takes values in the interval [0 .. 0,1]

Using fuzzy values (very important, important, not so important, etc.) facilitates the stakeholders’ task of attributing priorities to conflicting concerns.

Weights will be given to concerns with respect to the concern for which we have specified a composition rule. For example, with respect to *Mobility*, *Compatibility* can have a weight of 0.5, since accessing external services is not a fundamental issue in our system. In turn, *Availability* is very important for *Mobility* (a weight of 1.0), as without the system being available, we cannot offer mobility.

**Table 9.** Weighted (folded) contribution table

<i>Vis</i>	<i>Vis</i>							
<i>TIC</i>	+	<i>TIC</i>						
<i>Port</i>			<i>Port</i>					
<i>Mob</i>	+		+	<i>Mob</i>				
<i>ED</i>	+		+		<i>ED</i>			
<i>Cont</i>				+		<i>Cont</i>		
<i>Comp</i>	+	+		0,5			<i>Comp</i>	
<i>Avail</i>	+	+		1,0	1,0			<i>Avail</i>

**Resolve Conflicts.** The conflicts mentioned above for *Mobility* should not be too difficult to resolve, as the weights express priorities. If this was not the case negotiation would be needed among the stakeholders. Once all the conflicts have been resolved the specification is revised and recomposition carried out to identify any further conflicts.

**4.1.5 Specify Concern Dimensions**

Specification of a concern’s dimensions makes it possible to determine its influence on later development stages and identify its mapping onto a function, a decision or an aspect. The various concerns in our case study and their mappings and influences are shown in Table 10.

Consider our *Compatibility* concern. The requirements derived from this concern will influence parts of the system specification, architecture and design pertaining to requirements derived from other concerns constrained by it. They will also influence system evolution as change of the external services must be anticipated. The *Compatibility* concern will, however, map on to a function allowing visitors to connect to both hotel and ticket reservations. The *Mobility* concern, on the other

**Table 10.** Concern dimension specification

Concern	Influence	Mapping
Visitor	Specification, design, evolution	Function
Tourist inf. centre	Specification, design, evolution	Function
Portability	Specification, architecture, design, implementation	Decision
Mobility	Specification, architecture, design, implementation	Aspect
Electronic device	Specification, architecture, design	Function
Context	Architecture, design, implementation	Aspect
Compatibility	Specification, architecture, design, evolution	Function
Availability	Architecture	Decision

hand, will influence the specification, the type of architecture chosen and the design of the classes realising the requirements constrained by Mobility. It will map to an aspect at the design and implementation level because mobility properties cannot be encapsulated in a single class and will be otherwise spread across a number of classes.

## 5 Related Work

Multi-dimensional separation of concerns is supported by Hyperspaces [21] and Cosmos [20]. The Hyperspaces approach employs hyperslices as a decomposition mechanism where concerns are organised according to multiple dimensions, where each dimension is partitioned by concerns of the same type (e.g classes, functions). A hypermodule is a set of hyperslices together with a composition rule that specifies how the hyperslices are composed to form a more complex hyperslice. Our model can be seen as a specific instantiation of the hyperspaces model at the requirements level. Concerns in our model can be perceived as hyperslices while composition rules defining the projections can be seen as a specific instance of hypermodules. Cosmos is a concern-space modelling schema. Here a concern is any matter of interest in a system. A concern-space is an organised representation of concerns and their relationships. Similar to our work, Cosmos generalises the idea of a concern hyperspace (or hyperslice). It models concern-spaces through concerns, relationships and predicates. Concerns are classified as logical (representing concepts) and physical (representing elements of software systems). Some of the concerns and relationships e.g. physical ones are not relevant at the requirements level. Moreover, the projections of concerns on other concerns are not truly achieved.

Grundy proposes an aspect-oriented requirements engineering method targeted at component based software development [8]. The approach provides a categorisation of diverse aspects of a system that each component provides to end users or other components. A UML compliant approach to handle quality attributes (i.e. non-functional requirements) at the early stages of the development process is proposed in [14]. In both of these approaches, the separation of concerns is two-dimensional (i.e., functional and non-functional concerns (or aspects)). Moreover, the projections are limited from aspects to functional requirements.

In the Architecture Trade-off Analysis Method (ATAM) [11] various competing quality attributes and their interactions are characterised. This is achieved by building and maintaining both quantitative and qualitative models of these attributes. The

models are used as a basis to evaluate and evolve the architecture. The main focus of ATAM is on identifying the trade-off points at the architecture level. The work described in this paper focuses on identifying conflicting concerns in a uniform fashion and establishing critical trade-offs before the architecture is derived. Consequently, it is closer to the Twin Peaks model [15] which focuses on developing requirements and architectures in parallel in order to develop an early understanding of the system's technical feasibility, discover further requirements and constraints and evaluate alternative design solutions.

Theme/Doc [1] provides support for aspect-oriented analysis. Analysis is carried out by first identifying a set of actions in the requirements list which are, in turn, used to identify crosscutting behaviours. A theme is a collection of structures and behaviours that represent one feature. It is related to a concern in the work described here. While Theme/Doc is useful to identify themes in a requirements document, our approach complements this work by considering not only the identification of concerns, but also their specification and composition.

## 6 Conclusions and Future Work

In this paper, we have proposed a model to support multi-dimensional separation of concerns at the requirements level. This multi-dimensionality is achieved through a uniform treatment of both functional and non-functional properties. This is in direct contrast with existing RE approaches which typically focus on identifying the effects of non-functional requirements with reference to the functional requirements. Consequently, any broadly scoped influence of functional properties is not effectively dealt with.

The uniform treatment of concerns in our model makes it possible for us to define the projections of each concern on any set of concerns it relates to. By folding the resulting contribution matrix, we obtain a set of reflected projections which are then used for analysing the contribution of multiple concerns towards one particular concern. It is a difficult and complex task to derive such reflected projections otherwise.

The trade-off analysis and stakeholder negotiation supported by our RE model is based on a simple yet natural separation of concerns. This offers a powerful mechanism to identify influences of the various concerns in the system in a multi-dimensional fashion. This, in turn, supports better understanding of both crosscutting functional and non-functional requirements. Also, if a previously non-crosscutting set of requirements evolves to have a wider impact, the approach can easily deal with such a change through revision and recomposition of its projections. Any changes to relationships among concerns can be identified through requirements level impact analysis techniques [13].

Our future work will focus on developing case studies to further validate the proposed model and our set of concern specific actions and operators. In the near future we aim to incorporate the approach in our concern composition and decision support tool ARCADE which already provides support for the composition rules used for our case study. We are also interested in exploring the use of fuzzy logic for trade-

off analysis based on the weights we may give to concerns. This could help us identify a process to rank concerns by degree of importance in a system and use the result as a basis for incremental development.

## Acknowledgements

This work is supported by EPSRC Grant MULDRE (EP/C003330/1) and Portuguese FCT Grant SOFTAS (POSI/EIA/60189/2004).

## References

- [1] E. Baniassad and S. Clarke, "Theme: An approach for aspect-oriented analysis and design". In 26th International Conference on Software Engineering (ICSE), (Edinburgh, Scotland), 2004.
- [2] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*: Kluwer, 2000.
- [3] Å. Dahlstedt and A. Persson, "Requirements Interdependencies - Moulding the State of Research into a Research Agenda". The Ninth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2003), Klagenfurt/Velden, Austria, pp 71-80, 2003
- [4] A. Dardenne, A. Lamsweerde, and S. Fickas, "Goal-directed Requirements Acquisition", *Science of Computer Programming*, 20, pp. 3-50, 1993.
- [5] N. Davies, K. Cheverst, K. Mitchell, and A. Efrat, "Using and Determining Location in a Context-Sensitive Tour Guide", *IEEE Computer*, 34(8), pp. 35-41, 2001.
- [6] T. Elrad, R. Filman, and A. Bader (eds), "Theme Section on Aspect-Oriented Programming", *CACM*, 44(10), 2001.
- [7] A. Finkelstein and I. Sommerville, "The Viewpoints FAQ." *BCS/IEE Software Engineering Journal*, 11(1), 1996.
- [8] J. Grundy, "Aspect-Oriented Requirements Engineering for Component-based Software Systems", 4th IEEE International Symposium on Requirements Engineering, 1999, IEEE Computer Society Press, pp. 84-91.
- [9] I. Jacobson, *Object-Oriented Software Engineering - a Use Case Driven Approach*: Addison-Wesley, 1992.
- [10] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Software Engineering Institute Technical Report CMU/SEI-90-TR-21 1990.
- [11] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The Architecture Tradeoff Analysis Method", Proc. ICECCS, 1998, IEEE Computer Society Press, pp. 68-78.
- [12] A. Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour", 5th International Symposium on Requirements Engineering, 2001, IEEE Computer Society Press, pp. 249-261.
- [13] S. Lock and G. Kotonya, "An Integrated, Probabilistic Framework for Requirement Change Impact Analysis", *Australian Journal of Information Systems*, 6(2), 1999.
- [14] A. Moreira, J. Araújo, and I. Brito, "Crosscutting Quality Attributes for Requirements Engineering", In 14th International conference on Software Engineering and Knowledge Engineering (SEKE), 2002, ACM, pp. 167-174.

- [15] B. Nuseibeh, "Weaving Together Requirements and Architectures", *IEEE Computer*, 34(3), pp. 115-117, 2001.
- [16] A. Rashid, A. Moreira, and J. Araújo, "Modularisation and Composition of Aspectual Requirements", In International Conference on Aspect-Oriented Software Development (AOSD), 2003, ACM, pp. 11-20.
- [17] A. Rashid, P. Sawyer, A. Moreira, and J. Araújo, "Early Aspects: A Model for Aspect-Oriented Requirements Engineering", In International Conference on Requirements Engineering (RE), 2002, IEEE Computer Society Press, pp. 199-202.
- [18] P. Rayson, L. Emmet, R. Garside, and P. Sawyer, "The REVERE Project: Experiments with the application of probabilistic NLP to Systems Engineering", *Proc. NLDB 2000, LNCS 1959*, pp. 288-300.
- [19] I. Sommerville and P. Sawyer, *Requirements Engineering - A Good Practice Guide*: John Wiley and Sons, 1997.
- [20] S. M. Sutton and I. Rouvellou, "Modeling of Software Concerns in Cosmos", In International Conference on Aspect-Oriented Software Development (AOSD), 2002, ACM, pp. 127-133.
- [21] P. L. Tarr, H. Ossher, W. H. Harrison, and S. M. Sutton, "N Degrees of Separation: Multi-Dimensional Separation of Concerns", In International Conference on Software Engineering (ICSE), 1999, ACM, pp. 107-119.
- [22] S. Viller and I. Sommerville, "Social Analysis in the Requirements Engineering Process: From Ethnography to Method", In International Conference on Requirements Engineering (RE), 1998, IEEE Computer Society, pp. 6-13.
- [23] E. Yu, "Modelling Strategic Relationships for Process Reengineering": PhD Thesis, University of Toronto, 1995.



# Generating Transformation Definition from Mapping Specification: Application to Web Service Platform

Denivaldo Lopes<sup>1,2</sup>, Slimane Hammoudi<sup>1</sup>, Jean Bézivin<sup>2</sup>, and Frédéric Jouault<sup>2,3</sup>

<sup>1</sup> ESEO, France

<sup>2</sup> Atlas Group, INRIA and LINA, University of Nantes, France

<sup>3</sup> TNI-Valiosys, France

{dlopes, shammoudi}@eseo.fr

{jean.bezivin, frederic.jouault}@lina.univ-nantes.fr

**Abstract.** In this paper, we present in the first part our proposition for mapping specification and generation of transformation definition in the context of Model Driven Architecture (MDA). In the second part, we present the application of our proposition to Web Services platform. We propose a metamodel for mapping specification and its implementation as a plug-in for Eclipse. Once mappings are specified between two metamodels (e.g. UML and WSDL), transformation definitions are generated automatically using transformation languages such as Atlas Transformation Language (ATL). We have applied this tool to edit mappings between UML and Web Services. Then, we have used this mapping to generate ATL code to achieve transformations from UML into Web Services.

**Keywords:** Model Driven Architecture (MDA), Web Services, Tools for MDA.

## 1 Introduction

Recently, the OMG has proposed the Model Driven Architecture (MDA<sup>TM</sup>)<sup>1</sup> [1] to support the development of complex and large software systems providing an architecture with which:

- systems can evolve for meeting new requirements.
- old, current and new technologies can be harmonized.
- business logic is protected against the changes in technologies.
- legacy systems are integrated and harmonized with new systems.

In this approach, models are applied in all steps of development up to a target platform, providing source code, files of deployment and config, and so on. MDA proposes an architecture to address the complexity of software development and maintenance which has no precedents. It claims that software developers can create and maintain

---

<sup>1</sup> MDA<sup>TM</sup> is a trademark of the Object Management Group (OMG).

software artifacts with little effort. However, before this becomes a mainstream reality some issues in MDA approach need solutions such as *mapping specification* and *transformation definition* [2].

In this paper, we use the term *mapping* as a synonym for correspondence between the elements of two metamodels, while *transformation* is the activity of transforming a source model into a target model in conformity with the *transformation definition*. In our approach, a transformation definition is generated from a mapping specification. The distinction between mapping specification and transformation definition is detailed in later sections.

The objective of this paper is fourfold. First, to provide a precise definition of the concepts of mapping and transformation. Second, to provide a general metamodel for mapping specification in the context of MDA. Third, to present a tool based on Eclipse enabling the editing of mappings and the generation of transformation definition from mapping specifications. Fourth, to apply our tool to Web Service Platform.

This paper is organized in the following way. Section 2 is an overview of MDA. Section 3 presents our approach for mapping specification between two metamodels in the context of MDA. Section 4 illustrates our approach applied to Web Services. Section 5 shows the implementation of our proposed metamodel for mapping through a plug-in for Eclipse, and its application to Web Services. Finally, section 6 concludes this paper and presents the future directions of our research.

## 2 Overview

At the beginning of this century, software engineering needs to handle software systems that become larger and more complex than before. The object-oriented and component technology seems insufficient to provide satisfactory solutions to support the development and maintenance of these systems. To adapt to this new context, software engineering has applied an old paradigm, i.e. models, but with a new aspect, i.e. Model Driven Architecture (MDA).

Some ideas around the MDA approach are not new. For example, the generation of code from a model exists from the 80's, the transformation from models into a target platform was applied some time ago to the database domain (e.g. transformation from entity-relationship to relational-tables and SQL schema). However, the standardization of an approach based on models to enable the development and maintenance of software systems is a big advance in software engineering. The change from object-oriented and component paradigm to the model paradigm was inevitable and should be irreversible. However, this does not mean the end of the former, but the introduction of models as a supplementary layer to address the development of complex and large software systems. In fact, models are the top layer and the other paradigms are the bottom layer in the MDA approach.

We cannot yet advocate that the MDA approach will resolve all problems in software system development because some issues are not well settled such as mapping, transformation, semantic distance, traceability, and so on. However, several case studies have demonstrated that MDA is a potential solution and the future for developing software systems [3].

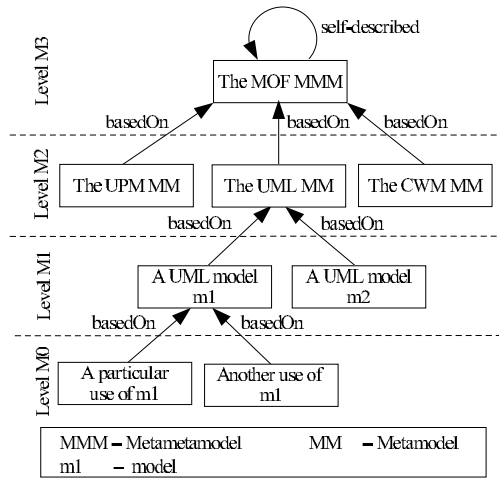


Fig. 1. Architecture with four meta-layers

### 2.1 The Architecture with Four Meta-layers

MDA is based on an architecture with four meta-layers [4]: metamodel, meta-model, model and information (i.e. an implementation of its model). Figure 1 presents the idea and the relationships between different levels of models. In this approach, everything is a model or a model element, and a high level of abstraction about a problem and its solution are provided.

In level M3, a metamodel is a well-formed specification for creating metamodels. In level M2, a metamodel is a well-formed specification for creating models. In level M1, a model is a well-formed specification for creating software artifacts. In level M0, an operational example of a model is the final representation of a software system. According to this architecture, we can envisage the existence of few metamodels such as MOF [4] and Ecore [5], several metamodels such as UML, UEMML [6] and EDOC [7], more models describing real life applications such as a travel agency, and finally infinite information such as the implementation of this travel agency model using Java. Here, it is important to pay attention to the existence of several metamodel languages, providing a Domain-Specific Language [8] or a general-purpose language (e.g. UML). In fact, the four layers are models. However, it is important to understand that each model level achieves a different goal in software development.

The development of software systems using MDA is based on the separation of concerns (e.g. business and technical concerns) which are afterwards transformed between them. So, business concerns are represented using Platform-Independent Model (PIM), and technical concerns are represented using Platform-Specific Model (PSM).

## 3 Mapping and Transformation

Nowadays, MDA suffers from a lack of agreement on terminology, especially concerning the concepts of mapping and transformation. In MOF QVT [2], mapping is defined

as *specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel*. In MDA distilled book [9], mapping is defined as *the application or execution of a mapping function in order to transform one model to another*, and mapping function is defined as *a collection of mapping rules that defines how a particular mapping works*. In both references and others discussed in [10], the concepts of mapping and transformation are not so clear, since these terms can refer to many different concepts. Moreover, they are usually defined without explicit distinction between them.

According to our vision, the concepts of mapping and transformation should be explicitly distinguished, and together could be involved in the same process that we denominate transformation process. In fact, in the transformation process, the mapping specification precedes the transformation definition. A mapping specification is a definition (as declarative as possible [11]) of the correspondences between metamodels (i.e. a metamodel for building a PIM and another for building a PSM). Transformation definition<sup>2</sup> contains a minute description to transform a model into another using a hypothetical or concrete transformation language. Hence, in our approach the transformation process of a PIM into a PSM can be structured in two stages: mapping specification and transformation definition. Finally, we define the term transformation as the manual or automatic generation of a target model from a source model, according to a transformation definition.

From a conceptual point of view, the explicit distinction between mapping specification and transformation definition remains in agreement with the MDA philosophy, i.e. the separation of concerns. Moreover, a mapping specification could be associated with different transformation definitions, where each transformation definition is based on a giving transformation definition metamodel.

Figure 2 illustrates the different concepts of MDA according to our vision where mapping specification is a mapping model, and transformation definition is a transformation model. In this figure, a mapping model is based on its metamodel, and it relates two metamodels (left and right). A transformation model is based on its transformation metamodel, and it is generated from a mapping model. A transformation engine takes a source model as input, and it executes the transformation program to transform this source model into the target model.

Several research projects have studied the mapping specification between metamodels [13] [14]. However, the ideas around mapping specification are not sufficiently developed to create efficient tools to enable automatic mappings in the context of MDA.

Nowadays, transformation languages are not yet very well explored to make choices about a standard transformation language such as desired by OMG [2]. In the next few years, the submitted propositions [15] [16] in response to QVT RFP might converge to a standard language, which will provide a new step forward in the evolution of MDA. However, wisdom tells us that one problem can be resolved using different solutions, but one solution for all problems does not exist. Thus, it is clear that this standard language

---

<sup>2</sup> In [12], transformation definition is *a set of transformation rules that together describes how a model in a source language can be transformed into a model in the target language*.

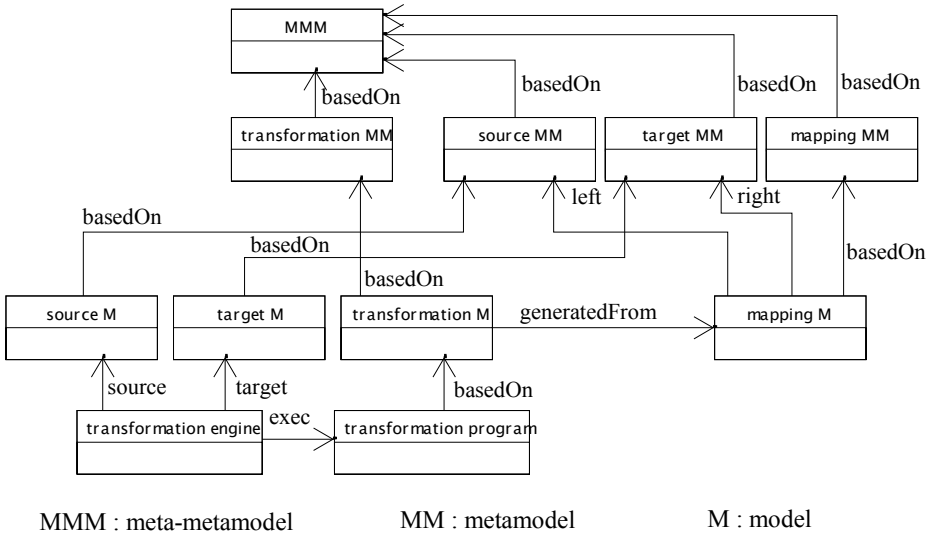


Fig. 2. Transformation process within MDA: from mapping to transformation

will not provide a sufficient solution for all types of model transformations around MDA. However, this will not be a limitation for applying MDA, because a transformation language is also a model, thus one transformation language can also be transformed into another transformation language. A priori, transformations between transformation languages seem unnecessary and unproductive. However, several examples such as Structured Query Language (SQL) (i.e. a standard query language for manipulating databases) have demonstrated that a standard is beneficial, because it establishes a unique and well-known formalism for understanding a problem and its solution. On the one hand, SQL provides a universal language for manipulating databases. On the other hand, SQL can be transformed into a proprietary language for execution into a database engine. A transformation from SQL into a proprietary language provides some benefits such as improved performance, reduction of memory-use, and so on. Making an analogy between SQL and a standard transformation language, we can expect that a standard transformation language can provide some benefits without imposing severe limitations.

Mapping and transformation have been studied for a long time ago in the database domain [11] [17]. However, they have taken another dimension with the sprouting of MDA. This does not mean that they are well-studied and ready to be applied in the MDA context. In fact, mapping specification and transformation definition are not yet an easy task. Moreover, tools to enable the automatic creation of mapping specification and automatic generation of transformation definition are still under development. Some propositions enabling the mapping specification have been based on heuristics [18] (for identifying structural and naming similarities between models) and on machine learning (for learning mappings) [19]. Other propositions enabling transformation definition have been based on graph theory [20] and compilers.

In this section, we start briefly presenting a foundation for mapping and afterwards we discuss our proposition for specifying mappings (i.e. correspondences between me-

tamodels). This approach for mapping is based on a metamodel and implemented as a tool on Eclipse. This tool provides mapping support that is a preliminary step before the generation of a transformation definition.

### 3.1 Foundation for Mapping Specification

A mapping specification can be formalized as follows:

Given  $M_1(s)/M_a$ ,  $M_2(s)/M_b$ , and  $C_{M \rightarrow M} / M_c$ , where  $M_1$  is a model of a system  $s$  created using the metamodel  $M_a$ ,  $M_2$  is a model of the same system  $s$  created using the metamodel  $M_b$ , and  $C_{M \rightarrow M}$  is the mapping between  $M_a$  and  $M_b$  created using the metamodel  $M_c$ , then a transformation can be defined as the function  $Transf(M_1(s)/M_a, C_{M \rightarrow M} / M_c) \rightarrow M_2(s)/M_b$ . In this section, we aim to detail  $C_{M \rightarrow M} / M_c$ . In general,  $M_a$ ,  $M_b$  and  $M_c$  are based on the same metamodel which simplifies the mapping specification. For now, we can define the mapping as  $C_{M \rightarrow M} \supseteq \{M_a \cap M_b\}$ , where  $\cap$  is a binary operator that returns the elements of  $M_a$  and  $M_b$  which have equivalent structure and semantics.

We can also represent a mapping as a set. So, given:

$$M_a = \{a_1, a_2, a_3, \dots, a_m\} \text{ and } M_b = \{b_1, b_2, b_3, \dots, b_n\}$$

Then,

$$C_{M \rightarrow M} = \{c_1, c_2, c_3, \dots, c_p\}$$

Where:

$$c_i = \{a_k, b_j\}$$

$$i = \{i \in N \mid 1 \leq i \leq p\}, k = \{k \in N \mid 1 \leq k \leq m\} \text{ and } j = \{j \in N \mid 1 \leq j \leq n\}.$$

In fact, models (i.e. in the general sense: models, metamodels and metametamodels) can be represented as sets. However, these sets are complex and heterogeneous, because their elements are classes, attributes, relationships, enumerations and datatypes. Thus, the creation of a mapping is not an easy task.

For clarity reasons, we divide elements of a metamodel into two categories: basic elements and relationships. Basic elements groups classes, attributes, enumerations and datatypes. Relationships relate classes. So,  $C_{M \rightarrow M}$  must satisfy the following requirements to be a complete mapping:

1. **Basic element preservation:** each basic element of  $M_a$  must verify one of the following requirements:
  - it corresponds to an equal ( $=$ ) basic element of  $M_b$ .
  - it corresponds to a set of basic elements of  $M_b$  that are similar ( $\cong$ )<sup>3</sup>.
  - it is part of a set of basic elements from  $M_a$  that together are similar to one basic element in  $M_b$ .
2. **Relationship preservation:**
  - each relationship in  $M_a$  must verify one of the following requirements:
    - it has a corresponding relationship in  $M_b$ .
    - it has a corresponding set of relationships of  $M_b$  that are similar ( $\cong$ ).
    - it can be implicitly preserved in  $M_b$  (i.e. through aggregating attributes or merging classes).

<sup>3</sup> As in [17], by similar, “we mean that they are related but we do not express exactly how”.

If  $M_b$  requires one basic element or relationship that can be deduced from two or more elements or relationships, respectively, from  $M_a$ , then the need for element and relationship preservation is satisfied.

If a mapping cannot satisfy the two requirements, then it is not complete, and the transformation definition is also not complete. Consequently, a target model generated from a source model using this transformation definition does not contain all the information of the source model.

The process of identifying and characterizing inter-relationships between metamodels is denoted *schema matching* [18]. In fact, *mapping* describes how two metamodels<sup>4</sup> are related to each other. So, *schema matching* results in a *mapping*. According to *model management algebra* [17], a *mapping* is generated using an operator called *match* which takes two metamodels as input and returns a *mapping* between them. We have adapted this operator as follow: given  $M_a$ ,  $M_b$  and  $C_M \rightarrow_M / M_c$ , the adapted *match operator* is formally defined as  $Match'(M_a, M_b) = C_M \rightarrow_M / M_c$ .

The identification of inter-relationships between two metamodels is generally based on the metamodel structure. A metamodel structure is a consequence of relationships between its elements. These relationships have some characteristics such as *kind*. Generally, five *relationship kinds* can relate one element to another element [14]: *Association*, *Contains*, *Has-a*, *Is-a*, *Type-of*. These relationships have been used for a long time in software engineering. For example, they are common in UML: *Association* is association; *Contains* is composition; *Has-a* is aggregation; *Is-a* is inheritance; *Type-of* relates a class that is a type of an attribute. These relationship kinds can be formalized as follow:

- **Association:**  $A(a, b)$  means  $a$  is associated with  $b$ .
- **Contains:**  $C(c, d)$  means container  $c$  contains  $d$ .
- **Has-a:**  $H(e, f)$  means  $e$  has an  $f$ .
- **Is-a:**  $I(g, h)$  means  $g$  is an  $h$ .
- **Type-of:**  $T(i, j)$  means  $i$  is a type of  $j$ .

In [14], the authors propose the following cross-kind-implications:

- if  $T(q, r)$  and  $I(r, s)$  then  $T(q, s)$ .
- if  $I(p, q)$  and  $H(q, r)$  then  $H(p, r)$ .
- if  $I(p, q)$  and  $C(q, r)$  then  $C(p, r)$ .
- if  $C(p, q)$  and  $I(q, r)$  then  $C(p, r)$ .
- if  $H(p, q)$  and  $I(q, r)$  then  $H(p, r)$ .

In [14], two models are considered equivalent “*if they are identical after all implied relationships are added to each of them until a fixed point is reached*”. Applying these relationship kinds and cross-kind-implications in MDA context, metamodels can be simplified and compared to find equivalences or similarities. Moreover, applying these principles and the operator  $Diff'$  (defined hereafter) based on the same principles presented in [17], the *semantic distance* [21] can be quantified. This operator  $Diff'$  takes a metamodel  $M_a$  and a mapping  $C_M \rightarrow_M / M_c$ , and returns a subset containing the elements of  $M_b$  that do not participate in the mapping. Formally,

<sup>4</sup> In our approach, we prefer to employ the term metamodel in the definition of the term mapping.

$Diff'(M_a, C_M \rightarrow M / M_c) = M_d$ , where  $M_d \subset M_b$ . We could quantify *semantic distance* using a numeral value, but as metamodels can be considered as a set, then we prefer to quantify *semantic distance* as a sub-set of  $M_b$ . In spite of [17], we do not consider the result of  $Diff'$  a sub-metamodel as expected, because we understand that this difference will return only fragments of a metamodel.

### 3.2 A Metamodel for Mappings

A metamodel for mapping must enable the specification of inter-relationships (i.e. correspondences) between the elements from two metamodels without modifying them. It should also provide support to handle different versions of a mapping.

Figure 3 presents a metamodel for mapping specification that meets these requirements.

In this metamodel, we consider that a mapping can be unidirectional or bidirectional. In unidirectional mapping, a metamodel is mapped into another metamodel. In bidirectional mapping, the mapping is specified in both directions. Thus, we prefer to call two metamodels in a mapping as left or right metamodel.

This metamodel for mapping presents the following elements:

- Element is a generalization for the other elements.
- Historic enables the explanation of the different choices taken for making the mapping. It has the date of the last update, a note, and the number of the last version, and a collection of Definitions.

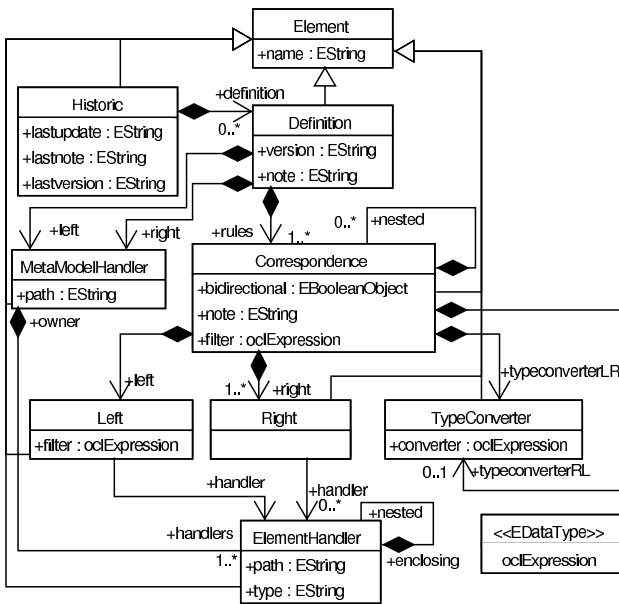


Fig. 3. Metamodel for Mapping Specification



- Definition is the main element and it contains all Correspondences between two metamodels (i.e. each correspondence has one left element and many right elements).
- Correspondence is used to specify the inter-relationship between two or more elements, i.e. one left and one or more right elements. The correspondence has a filter that is an OCL expression. When `bidirectional` is false, a mapping is unidirectional (i.e. left to right), and when it is true it is bidirectional (i.e. in both directions). It has two `TypeConverters` identified by `typeconverterRL` and `typeconverterLR`. `typeconverterRL` enables the conversion of the elements from a right metamodel into the elements from a left metamodel. `typeconverterLR` enables the conversion of the elements from a left metamodel into the elements from a right metamodel. Often we need to specify only the `typeconverterLR`.
- Left is used to identify the left element of a mapping.
- Right is used to identify the right elements of a mapping.
- `MetaModelHandler` is used to navigate into a metamodel. It has the information necessary for accessing a metamodel participating in a mapping. A mapping is itself a model, and it must not interfere with the two metamodels being mapped.
- `ElementHandler` enables access to the elements being mapped without changing them.
- `TypeConverter` enables the type casting between a left and a right element. If one element of a left metamodel and another element of a right metamodel are equals, then the mapping is simple and direct. However, if one element of a left metamodel and another element of a right metamodel are similar, then the mapping is complex and it is achieved using type converter, i.e. a complex expression to adapt a left element to a right element.

### 3.3 A Graphical Notation for Mapping

In order to simplify the mapping task, the description of a mapping specification based on our proposed metamodel for mapping should have a simplified graphical notation such as depicted in figure 4.

According to figure 4, some metamodel elements have a graphical representation. `Historic` is represented using a table. `MappingDefinition` is represented using a form that has correspondences. `Correspondence` is represented by a circle. `Left` is represented by a single arrow. `Right` is represented by a double arrow.

## 4 Applying MDA for Web Services

Nowadays, MDA is not sufficiently developed and experimented. In our research, we develop the MDA approach and we use Web Services as a target platform to experiment it.

Web Services have been introduced to resolve the problem of interoperability on the Internet. In fact, Web Services were created using the standards suitable for Internet. Consequently, they are more adapted to Internet than previous solutions such as

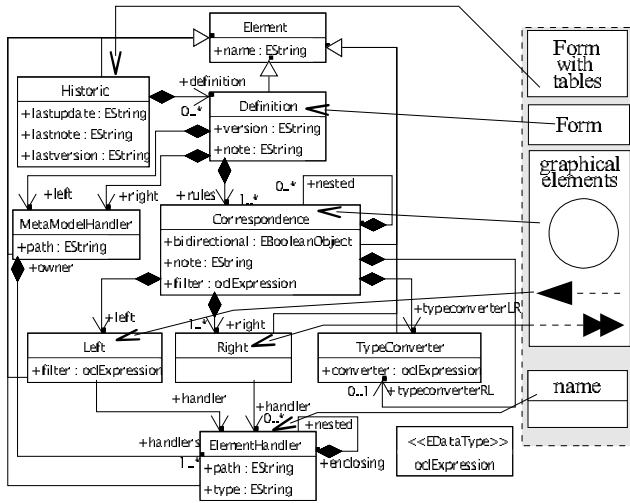


Fig. 4. Metamodel for mapping and its graphical notation

CORBA, Java RMI and EJB. However, they only provide support to the development of software systems in low level, thus the interoperability is only guaranteed in the level of implementation. MDA approach proposes an interoperability in the level of models which seems to be a promising solution. In addition, it provides mobility, i.e. a same business model can be implemented on different target platforms.

4.1 Web Services

The concept of services was introduced before Web Services. In fact, this concept has been used for a long time by OSF’s Distributed Computing Environment (DCE), OMG’s CORBA, Sun’s Java RMI, and Microsoft’s Distributed Component Object Model (DCOM)<sup>5</sup>. A service is an abstraction of programs, business process, and other artifacts of software defined in terms of what it does.

Service Oriented Architecture (SOA) [22] describes how a system composed of services can be built. Developing applications on SOA requires the adoption of a service-oriented design<sup>6</sup> which is based on the requirements determined in the strategy and business process levels.

Figure 5 shows the main SOA elements. An AgentProvider has Services. These Services are described through a meta-data representation, i.e. ServiceDescription. Afterwards, the AgentProvider stores information about its Services in a Registry. An AgentRequester searches in the Registry for a specific service according to a determined criterion. The Registry returns information about a desired service. The AgentRequester finds the meta-data about this service and uses it to exchange messages with the service. According to this figure, Universal Description,

<sup>5</sup> The actual COM+ is descendant of DCOM.

<sup>6</sup> Web Services is not inherently compliant to service-oriented design and to SOA.

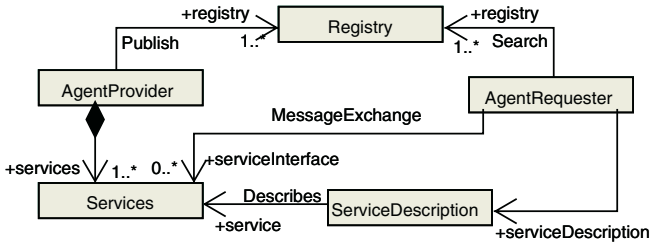


Fig. 5. Service Oriented Architecture (fragment)

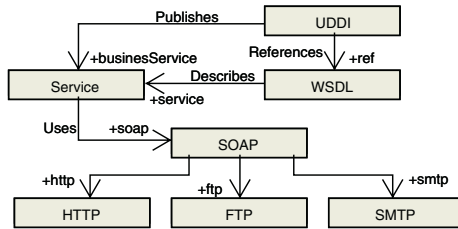


Fig. 6. Web Services (main technologies)

Discovery, and Integration (UDDI) [23] implements the Registry. Web Service Description Language (WSDL) [24] implements the ServiceDescription. Services use Simple Object Access Protocol (SOAP) [25] as a communication protocol, and SOAP uses HTTP or FTP or SMTP as transport protocol. Figure 6 presents the main technologies of Web Services and their relationships.

However, some issues related to Web Services still need solutions such as service composition, security and availability. Web Service composition can be static or dynamic. In a static composition, the services are determined and composed in the design time, while in the dynamic composition, the services are determined and composed at runtime. Some languages were proposed to take into account the service composition such as Business Process Execution Language for Web Services (BPEL4WS) [26].

#### 4.2 MDA and Web Services: Mapping Specification

Web Services are the main target platform used in our experiments, and B2B applications are our privileged domain. In this paper, we present the application of our tool to map UML into Web Services, and afterwards to generate the corresponding transformation definition with Atlas Transformation Language (ATL). In order to simplify the presentation of this paper, we only show experiments with UML, WSDL [24] and BPEL4WS [26].

Figure 7 depicts a mapping specification from UML into WSDL. This representation is based on the graphical notation presented in section 3.3. For the moment, we have used this graphical notation to illustrate mappings, but we aim to introduce it in the next version of our plug-in.

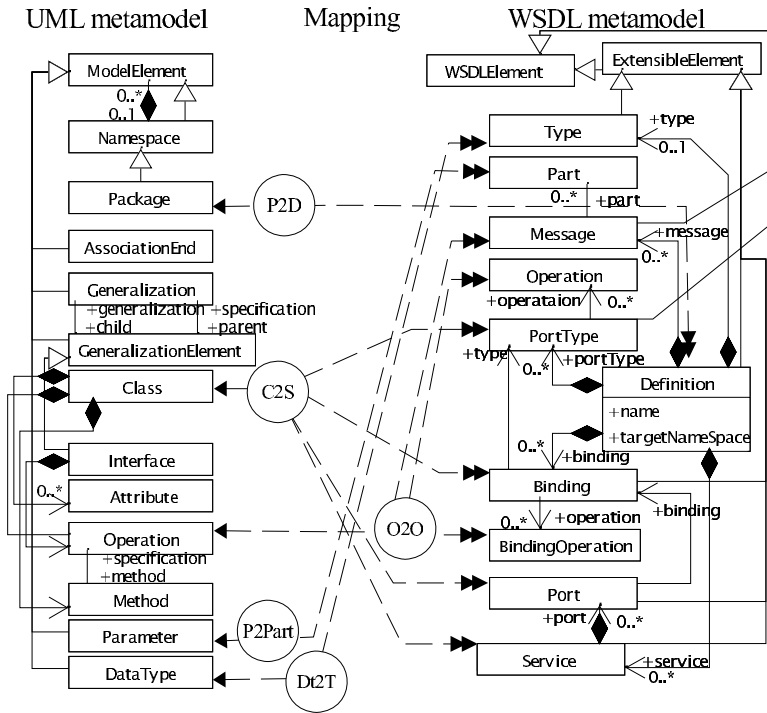


Fig. 7. A mapping from UML into WSDL (fragment)

According to figure 7, P2D maps Package into Definition, C2S maps Class into Service, Port, Binding and PortType<sup>7</sup>, and so on. It is important to note that C2S is a mapping one-to-many, i.e. it takes one element and maps it into many elements.

## 5 Tool for Mapping

A tool for supporting mappings between metamodels should provide the following characteristics:

- importation of pre-existing metamodels from XMI file.
- graphical visualization of the mapping model and metamodels.
- edition of the mapping model.
- verification of conformity between mapping model and its metamodel.
- another simplified representation of a mapping model such as textual representation.
- navigation between the metamodels that are being mapped.
- semi-automatic matching.

<sup>7</sup> PortType was renamed to Interface in WSDL 1.2

- generation of a transformation definition from a mapping specification (i.e. mapping model).
- exportation of a mapping model using XMI file.

Our proposed tool supports all these characteristics, except the *semi-automatic matching* which is the next step for its improvement.

### 5.1 Mapping Modeling Tool (MMT)

Figure 8 shows our plug-in for Eclipse denominated Mapping Modeling Tool (MMT) that supports the mapping modeling. MMT presents a first metamodel on the left side, a mapping model in the center, and a second metamodel on the right. In this figure, the UML metamodel (fragment) is mapped into a WSDL metamodel (fragment). At the bottom, the property editor of mapping model is shown. A developer can use this property editor to set the properties of a mapping model.

Before specifying mapping using our tool, we need to create metamodels based on Ecore [5]. Some tools support the editing of a metamodel based on Ecore such as Omondo [27] or the Ecore editor provided with EMF [5]. The application of our tool using UML and WSDL metamodel can be explained in the following steps:

1. We created a project in eclipse and we imported the UML and WSDL metamodel into this project.

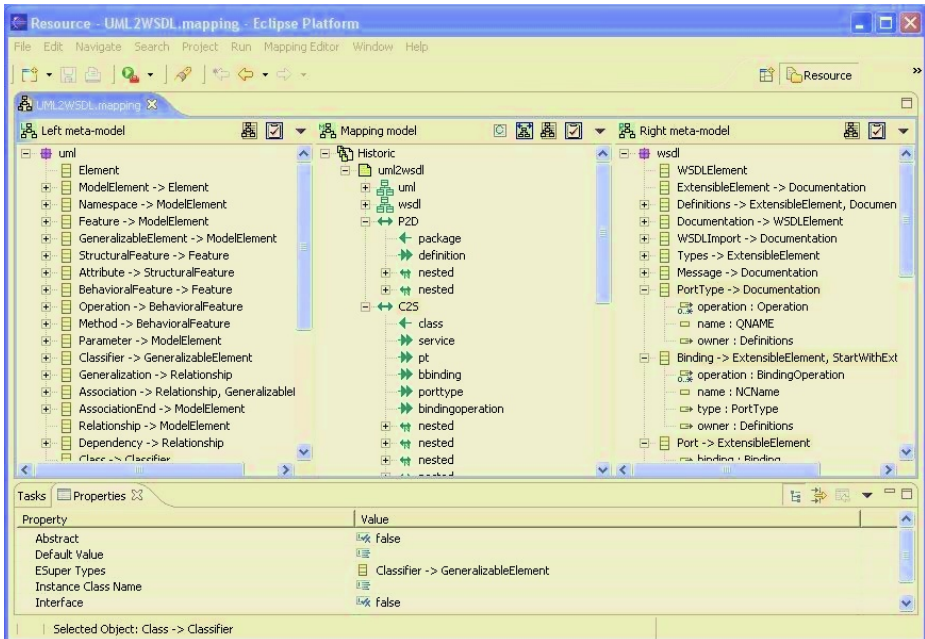


Fig. 8. Applying the tool to specify a mapping from UML into WSDL

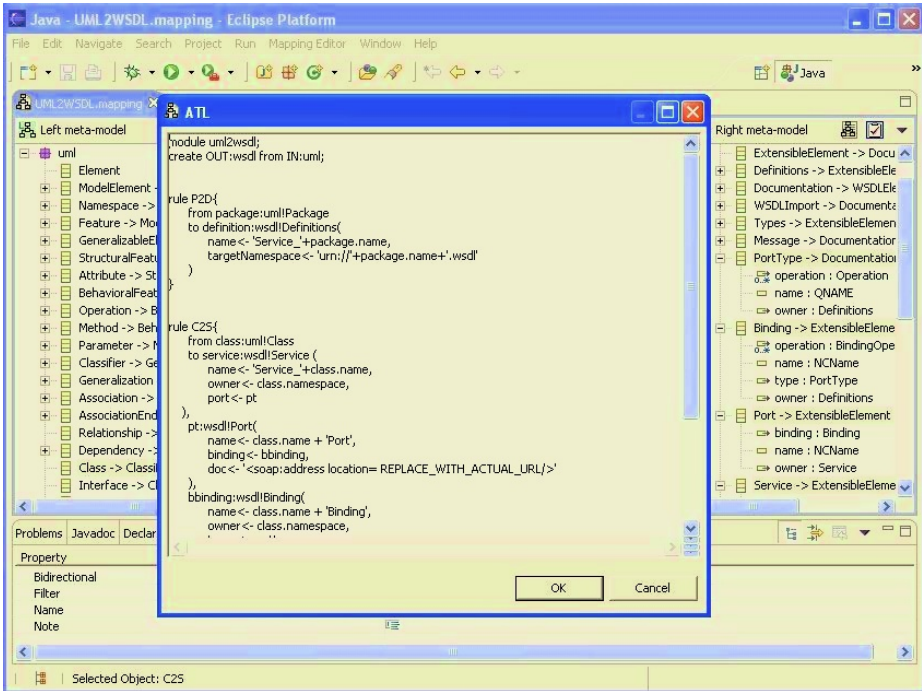


Fig. 9. The generated ATL code to transform UML into WSDL

2. We used a wizard to create a mapping model. In this step, we chose the name for the mapping model, the encoding of the mapping file (e.g. Unicode and UTF-8), the files of metamodel in the format XMI.
3. The UML and WSDL metamodels are loaded from the XMI files, and the mapping model is initially created, containing the elements *Historic*, *Definition*, and *left* and *right* *MetamodelHandlers*. For each *MetamodelHandler* are also created *ElementHandlers* that are references to the elements of the corresponding metamodel.
4. We edit the mapping model. First, we define the inter-relationships between the metamodels creating *Correspondences* between their elements. Second, we create for each *Correspondence* nested *Correspondences*. Third, for each nested correspondence, we create one *Left* and one or more *Right* elements. In addition, each *Left* and *Right* element has a *ElementHandler*. If it is necessary, the *TypeConverter* is created to explicit the casting between two mapped elements.
5. The mapping model can be validated according to its metamodel, and it can be used to generate a transformation definition (e.g. using ATL language).

According to figure 8, C2S maps *Class* into *Service*, *Port*, *Binding* and *PortType*.

MMT can generate transformation definition from a mapping model. For the moment, we have implemented a generator for ATL[28]. The resulting code in ATL of the

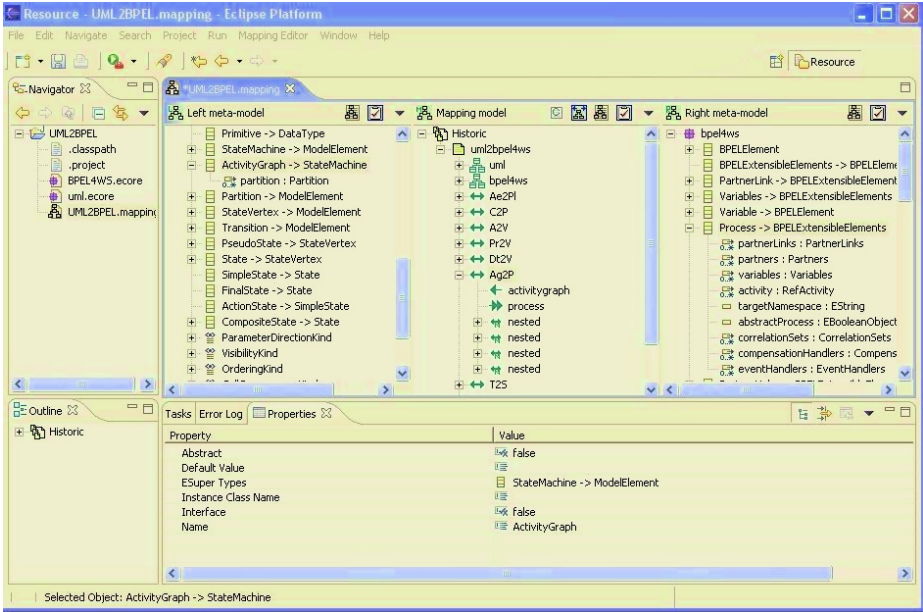


Fig. 10. Applying the tool to specify a mapping from UML into BPEL4WS

mapping between UML (fragment) and this WSDL metamodel is presented in figure 9. In this figure, a fragment of an ATL code is presented as `module uml2wsdl1; ...` and the rule C2S.

Figure 10 depicts the mapping model from UML into BPEL4WS [26] using our proposed tool. In this figure, Ag2P maps ActivityGraph into Process, Dt2V maps DataType into Variable, and so on. Since this mapping model is complete, MMT can generate the ATL code to realize transformations.

In this experiment, the ATL code was generated on the basis of the mapping model. This proposed tool left the developer free to think only at the point of the mapping between two metamodels, helping him to specify how the metamodels can be inter-related. Afterwards, it generated the code to transform models.

## 6 Conclusion

In this paper, we have discussed the MDA approach providing a detailed description of transformation process, distinguishing mapping and transformation. We have proposed a metamodel for mapping and a tool to support mappings. To illustrate our tool, we have specified mappings between UML as PIM and Web Services as PSM.

The *schema matching* was not yet integrated in our plug-in, because, at this stage, we are more interested in addressing the creation of mappings driven by models.

Some formalisms have been simplified and do not distinguish model, metamodel and metametamodel. Here, we have explicitly differentiated all model levels, because

a model can be created using different metamodels. Moreover, the diffusion of MOF, Ecore and DSL will stimulate an increase in available metamodels.

In future research, we will develop further the graphic representation (discussed in section 3.3) and the *schema matching* in order to integrate them also into our plug-in for Eclipse.

## Acknowledgments

The work described in this paper was partly financed by “Conseil Général de Maine-et-Loire”, through a fellowship provided to Denivaldo Lopes.

## References

1. OMG: Model Driven Architecture (MDA)- document number ormsc/2001-07-01. (2001)
2. OMG: Request for Proposal: MOF 2.0 Query/Views/Transformations RFP. (2002)
3. Middleware Company: Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach. Technical report, The Middleware Company (2003)
4. OMG: Meta Object Facility(MOF) Specification. (2002) Version 1.4.
5. Eclipse Tools Project: Eclipse Modeling Framework (EMF) version 2.0. (2004)
6. UEML.org: Unified Enterprise Modeling Language (UEML) (2003) Available at <http://www.ueml.org>.
7. OMG: UML Profile for Enterprise Distributed Object Computing Specification. (2002)
8. Cook, S.: Domain-Specific Modeling and Model Driven Architecture. MDA Journal (2004) 1–10
9. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. 1st edn. Addison-Wesley (2004)
10. Favre, J.M.: Towards a Basic Theory to Model Driven Engineering. UML 2004 - Workshop in Software Model Engineering (WISME 2004) (2004)
11. Velegrakis, Y., Miller, R.J., Popa, L.: Mapping Adaptation under Evolving Schemas. Proceedings of the 29th VLDB Conference (2003) 584–595
12. Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. 1st edn. Addison-Wesley (2003)
13. Caplat, G., Sourrouille, J.L.: Model Mapping in MDA. Workshop in Software Model Engineering (WISME2002) (2002)
14. Pottinger, R.A., Bernstein, P.A.: Merging Models Based on Given Correspondences. Proceedings of the 29th VLDB Conference (2003) 826–873
15. DSTC, IBM, CBOP: MOF Query / Views / Transformations - Second Revised Submission. (2004) ad/2004-01-06.
16. QVT-Merge Group: Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10). (2004) Available at <http://www.omg.org/docs/ad/04-04-01.pdf>.
17. Bernstein, P.A.: Applying Model Management to Classical Meta Data Problems. Proceedings of the Conference on Innovative Data Systems Research (CIDR 2003) (2003)
18. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal **10** (2001) 334–350
19. Martin S. Lacher, G.G.: Facilitating the Exchange of Explicit Knowledge through Ontology Mappings. 14th International FLAIRS Conference (2001) 21–23



20. Agrawal, A., Levendovszky, T., Sprinkle, J., Shi, F., Karsai, G.: Generative Programming via Graph Transformation in the Model-Driven Architecture. OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture (2002)
21. Bézivin, J., Hammoudi, S., Lopes, D., Jouault, F.: Applying MDA Approach for Web Service Platform. 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004) (2004) 58–70
22. W3C: Web Services Architecture (WSA). (2004)
23. UDDI.ORG: Universal, Description, Discovery and Integration (UDDI) Version 3.0. (2002)
24. W3C: Web Services Description Language (WSDL) 1.1. (2001)
25. W3C: Simple Object Access Protocol (SOAP) 1.1. (2001)
26. Andrews, T., Curbera, F., Dholakia, H., Golland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services (BPEL4WS) version 1.1. (2003)
27. Omondo: Omondo Eclipse UML. (2004) Available at <http://www.omondo.com>.
28. Bézivin, J., Dupé, G., Jouault, F., Pitette, G., Rougui, J.E.: First Experiments with the ATL Model Transformation Language: Transforming XSLT into XQuery. 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture (2003)

# A General Approach to the Generation of Conceptual Model Transformations

Nikolaos Rizopoulos and Peter M<sup>c</sup>Brien

Dept. Computing, Imperial College London, London SW7 2AZ  
{nr600, pjm}@doc.ic.ac.uk  
<http://www.doc.ic.ac.uk/automed>

**Abstract.** In data integration, a Merge operator takes as input a pair of schemas in some conceptual modelling language, together with a set of correspondences between their constructs, and produces as an output a single integrated schema. In this paper we present a new approach to implementing the Merge operator that improves upon previous work by considering a wider range of correspondences between schema constructs and defining a generic and formal framework for the generation of schema transformations. This is used as a basis for deriving transformations over high level models. The approach is demonstrated in this paper to generate transformations for ER and relational models.

## 1 Introduction

Initial research into data integration [1, 13] was concerned with the type of transformations that can be performed on the data source schemas [12, 24], while more recent research has focused on schema matching [15, 14, 8], *i.e.* identifying correspondences and semantic relationships between schema constructs. The process of **model management** incorporates the above by providing operators such as Match, Merge, *etc* for schemas [2]. In this paper, we are not concerned with the Match operator, which produces a set of correspondences between the schema constructs, but focus on the Merge operator, that takes as input two schemas, together with the result of Match, and produces as output a single integrated schema.

In [16, 6], schemas in a high level conceptual modelling language (such as ER, Relational, ORM, *etc*) are modelled in a nested **hypergraph data model (HDM)** [25, 22, 23]. We base our approach to implementing the Merge operator on determining how semantic relationships between nodes and edges in the HDM will cause transformations on the HDM to be generated, which can be mapped to BAV transformations [23, 18] on the high level modelling language. Based on these foundations, we provide a generic framework that can be used for merging schemas irrespective of the high-level modelling language used to represent them. This works by using the semantics of the high level modelling language to determine which of the low level rules may be applied.

Our methodology has the advantage of providing a generic solution to the problem of generating transformations, since it relies on the underlying graphical properties of data modelling languages, and not on the specific modelling language that is being used in a particular **universe of discourse (UoD)**. In addition, it deals with with a variety of semantic relationships — subsumption, disjointness, intersection, and equivalence —

between schema constructs, while most existing merging techniques deal with just the equivalence semantic relationship [3, 19]. As a result, our approach does not only merge schemas but it also improves them to remove any structural redundancy.

The structure of this paper is as follows. Section 2 describes the types of semantic relationships we use as input to our Merge operator. Section 3 gives an example of how a systems integrator might use a given set of semantic relationships to perform *manually* data integration with BAV transformations. An informal justification of how the BAV transformations are derived from the semantic relationships is given, and this acts as a motivation for the generic rules. Section 4 reviews details of the HDM, and illustrates how it is used to represent the ER and relational schemas we use in this paper. Then Section 5 shows how a set of generic rules operating over the HDM may be used to generate transformations in the higher level modelling languages from semantic relationships, and in particular the transformations of Section 3. Related work is in Section 6 and our summary and conclusions are found in Section 7.

## 2 Semantic Relationships

Various types of semantic relationships between schema constructs have been defined in the literature. We adopt similar relationship definitions to [12], except for **disjointness**. The four types of semantic relationship between schema constructs  $A, B$  are based on the comparison of their intensional domains  $D_i(A), D_i(B)$ , *i.e.* the set of real world entities associated with the constructs. The relationships are:

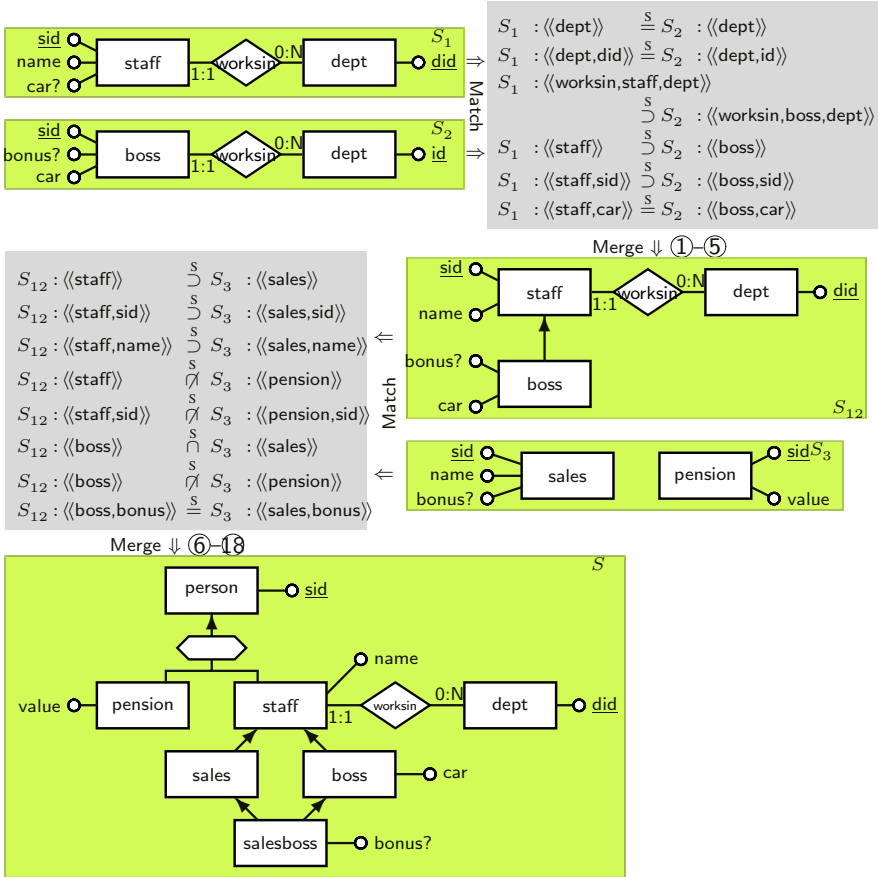
1. **equivalence**: Two schema constructs  $A$  and  $B$  are equivalent,  $A \stackrel{s}{=} B$ , iff  $D_i(A) = D_i(B)$
2. **subsumption**: Schema construct  $A$  subsumes schema construct  $B$ ,  $B \stackrel{s}{\subset} A$ , iff  $D_i(B) \subset D_i(A)$
3. **intersection**: Two schema constructs  $A$  and  $B$  are intersecting,  $A \overset{s}{\cap} B$ , iff  $D_i(A) \cap D_i(B) \neq \emptyset, \exists C : D_i(A) \cap D_i(B) = D_i(C)$
4. **disjointness**: Two schema constructs  $A$  and  $B$  are disjoint,  $A \overset{s}{\not\cap} B$ , iff  $D_i(A) \cap D_i(B) = \emptyset, \exists C : D_i(A) \cup D_i(B) \subseteq D_i(C)$

It is important to notice that construct  $C$  in the definition of intersection and disjointness may or may not exist in the schemas. The notation  $\exists C : \textit{condition}$  means that there is a real-world concept in the domain of the data source examined, that can be represented by an existing or non-existing schema construct  $C$  that satisfies the *condition*.

## 3 Motivating Examples of Integration

We now present two examples of data integration, where the examples differ *only* in the modelling language being used, and not in the UoD being considered. The examples will illustrate the intuition of how schema matching performed between data sources drives the integration process and leads to integration rules. Of course the integration rules necessarily differ in detail according to the data modelling language being used,

but they are triggered by the same conditions, they have common objectives and they perform analogous schema transformations.



**Fig. 1.** Three ER models being integrated. The ER model has key attributes underlined, and optional attributes followed by a question mark. Generalisations are indicated by hexagons, and dictate that their sub-entity classes are disjoint

### 3.1 ER Model Integration

Figure 1 illustrates a process where three ER schemas are integrated. First  $S_1^{er}$  and  $S_2^{er}$  are compared, and a set of semantic relationships is produced by Match. These relationships input into Merge, which integrates  $S_1^{er}$  and  $S_2^{er}$  into schema  $S_{12}^{er}$ . We then match  $S_{12}^{er}$  with  $S_3^{er}$ , producing another set of semantic relationships, which are then used to form the final global schema  $S_g^{er}$ . We will use BAV to specify the transformations necessary during data integration [18, 4], and adopt the three step conform, merge, restructure approach to schema integration [1]. Starting with integrating  $S_1^{er}$  and  $S_2^{er}$ ,

during the conform phase, the fact that  $\langle\langle\text{dept},\text{did}\rangle\rangle$  attribute in  $S_1^{er}$  is equivalent to the  $\langle\langle\text{dept},\text{id}\rangle\rangle$  attribute in  $S_2^{er}$  causes one to be renamed as the other:

①  $\text{renameAttribute}(\langle\langle\text{dept},\text{id}\rangle\rangle, \langle\langle\text{dept},\text{did}\rangle\rangle)$

During the merging phase, the fact that the concept of  $\langle\langle\text{boss}\rangle\rangle$  in  $S_2^{er}$  is subsumed by  $\langle\langle\text{staff}\rangle\rangle$  in  $S_1^{er}$  causes a subset relation to be introduced between the two entities:

②  $\text{addSubset}(\langle\langle\text{staff},\text{boss}\rangle\rangle)$

During the restructuring phase, we remove any redundancy that exists between the schemas. Since  $\langle\langle\text{worksin},\text{boss},\text{dept}\rangle\rangle$  in  $S_2^{er}$  is subsumed by  $\langle\langle\text{worksin},\text{staff},\text{dept}\rangle\rangle$  in  $S_1^{er}$ , we can delete the former construct without losing information in transformation ③. The fact that we are not losing information is verified by supplying a query that restores the extent of the construct we are deleting. Here we use a list comprehension [7] based language called IQL [11] used in the AutoMed system [5]. The expression in ③ states that we take those  $\{x\}$  values in entity  $\langle\langle\text{boss}\rangle\rangle$ , and then take those  $\{x, y\}$  found in relationship  $\langle\langle\text{worksin},\text{staff},\text{dept}\rangle\rangle$  with the same  $x$  value, and hence find those values of the  $\text{worksin}$  relationship that are associated to the boss entity. Also note that the rough semantics of each transformation is that the extent of the scheme in the first argument can be derived from the query that is second argument. If the first argument is a constraint, then it has no extent, and hence there is no second argument. Similarly, since  $\langle\langle\text{boss},\text{sid}\rangle\rangle$  is subsumed by  $\langle\langle\text{staff},\text{sid}\rangle\rangle$ , we eliminate  $\langle\langle\text{boss},\text{sid}\rangle\rangle$  in transformation ④. The fact that  $\langle\langle\text{staff},\text{car}\rangle\rangle$  and  $\langle\langle\text{boss},\text{car}\rangle\rangle$  are equivalent means that we should eliminate  $\langle\langle\text{staff},\text{car}\rangle\rangle$  in ⑤ since it is the less specific case of the car attribute, and can state as the IQL query that its values were all those instances of  $\langle\langle\text{boss},\text{car}\rangle\rangle$ .

③  $\text{deleteRelationship}(\langle\langle\text{worksin},\text{boss},\text{dept},1:1,0:\text{N}\rangle\rangle,$

$\{ \{x, y\} \mid \{x\} \leftarrow \langle\langle\text{boss}\rangle\rangle; \{x, y\} \leftarrow \langle\langle\text{worksin},\text{staff},\text{dept}\rangle\rangle \}$ )

④  $\text{deleteAttribute}(\langle\langle\text{boss},\text{sid},\text{key}\rangle\rangle, \{ \{x, y\} \mid \{x\} \leftarrow \langle\langle\text{boss}\rangle\rangle; \{x, y\} \leftarrow \langle\langle\text{staff},\text{sid}\rangle\rangle \})$

⑤  $\text{deleteAttribute}(\langle\langle\text{staff},\text{car},\text{null}\rangle\rangle, \langle\langle\text{boss},\text{car}\rangle\rangle)$

The resulting  $S_{12}^{er}$  is an integration of  $S_1^{er}$  and  $S_2^{er}$  that obeys one important feature of the integration rules of the framework: that **pathway**  $S \rightarrow S'$  of transformations from schema  $S$  to  $S'$  satisfy the **relationship preservation property (RPP)**. The RPP states that if the reverse pathway  $P' = S' \rightarrow S$  is followed, then the relationships initially existing in  $S$  are still true, *i.e.* the semantic relationships between the constructs are preserved. Implicitly, this means that the intentional domains of the constructs are not affected by the rules, *i.e.* they do not cause any real-world entity loss nor gain. The RPP is ensured by the fact that all add and delete transformations in the pathway ①–⑤ that add or delete constructs that have an associated **extent** (*i.e.* set of values) are supplied with queries that fully define that extent in terms of other constructs in the schema.

Integration now proceeds to match  $S_{12}^{er}$  with  $S_3^{er}$ . Since no naming conflicts are found, we proceed directly to merging phase. The fact that there is a intersection relationship between  $\langle\langle\text{sales}\rangle\rangle$  of  $S_3^{er}$  and  $\langle\langle\text{staff}\rangle\rangle$  of  $S_{12}^{er}$  means we can introduce a common subset entity  $\langle\langle\text{salesboss}\rangle\rangle$  by transformations ⑦–⑨. The IQL expression in ⑦ ensures that the new entity has instances that appear in both  $\langle\langle\text{sales}\rangle\rangle$  and  $\langle\langle\text{staff}\rangle\rangle$ , and this is also explicitly stated in the schema structure by the two subset constructs added by ⑧ and ⑨. The fact that there is a disjointness relationship between  $\langle\langle\text{pension}\rangle\rangle$  and  $\langle\langle\text{staff}\rangle\rangle$  means we can introduce a generalisation of them in the form of the  $\langle\langle\text{person}\rangle\rangle$  entity with transformations ⑩ and ⑪. In ⑩ the IQL append operator  $++$  is used to append all values of  $\langle\langle\text{pension}\rangle\rangle$  to those of  $\langle\langle\text{staff}\rangle\rangle$ .

- ⑥ addSubset(⟨⟨staff,sales⟩⟩)
- ⑦ addEntity(⟨⟨salesboss⟩, [{x} | {x} ← ⟨⟨sales⟩⟩; {x} ← ⟨⟨boss⟩⟩]⟩)
- ⑧ addSubset(⟨⟨sales,salesboss⟩⟩)
- ⑨ addSubset(⟨⟨boss,salesboss⟩⟩)
- ⑩ addEntity(⟨⟨person⟩, ⟨⟨pension⟩ ++ ⟨⟨staff⟩⟩⟩)
- ⑪ addGeneralisation(⟨⟨person,pension,staff⟩⟩)

During restructuring, transformations ⑫–⑭ perform attribute specialisation, combining the equivalent ⟨⟨sales,bonus⟩⟩ and ⟨⟨boss,bonus⟩⟩ into ⟨⟨salesboss,bonus⟩⟩. Then ⑮–⑰ perform attribute generalisation, combining ⟨⟨pension,sid⟩⟩ and ⟨⟨staff,sid⟩⟩. Finally ⑱ removes the redundant ⟨⟨sales,name⟩⟩ that is subsumed by ⟨⟨staff,name⟩⟩. The result of these transformations is the final integrated schema  $S_g^{er}$  in Figure 1.

- ⑫ addAttribute(⟨⟨salesboss,bonus,null⟩⟩, ⟨⟨sales,bonus⟩⟩)
- ⑬ deleteAttribute(⟨⟨sales,bonus,null⟩⟩, ⟨⟨salesboss,bonus⟩⟩)
- ⑭ deleteAttribute(⟨⟨boss,bonus,null⟩⟩, ⟨⟨salesboss,bonus⟩⟩)
- ⑮ addAttribute(⟨⟨person,sid,key⟩⟩, ⟨⟨pension,sid⟩⟩ ++ ⟨⟨staff,sid⟩⟩)
- ⑯ deleteAttribute(⟨⟨pension,sid,key⟩⟩, [{x,y} | {x,y} ← ⟨⟨person,sid⟩⟩; {x} ← ⟨⟨pension⟩⟩])
- ⑰ deleteAttribute(⟨⟨staff,sid,key⟩⟩, [{x,y} | {x,y} ← ⟨⟨person,sid⟩⟩; {x} ← ⟨⟨staff⟩⟩])
- ⑱ deleteAttribute(⟨⟨sales,name,nonnull⟩⟩, [{x,y} | {x,y} ← ⟨⟨staff,name⟩⟩; {x} ← ⟨⟨sales⟩⟩])

### 3.2 Relational Model Integration

Figure 2 shows the integration of three relational schemas  $S_1^{rel}$ ,  $S_2^{rel}$  and  $S_3^{rel}$  that are equivalent to the schemas  $S_1^{er}$ ,  $S_2^{er}$  and  $S_3^{er}$ . However, due to differences in the modelling language, the semantic relationships that are identified, and the integration transformations required are different from those discussed for the ER integration above. The final integrated schema  $S_g^{rel}$  is almost equivalent to the final  $S_g^{er}$  in Figure 1; the difference in semantics being that the relational model is unable to express the disjointness of ⟨⟨staff⟩⟩ and ⟨⟨pension⟩⟩ that is represented by the ER generalisation hierarchy. Our discussion of the relational integration focuses on comparing it with the ER integration, and stating why it is different. First in integrating  $S_1^{rel}$  and  $S_2^{rel}$ , the conforming phase has a transformation analogous to ①:

- ⑲ renameColumn(⟨⟨dept,id⟩⟩, ⟨⟨dept,did⟩⟩)

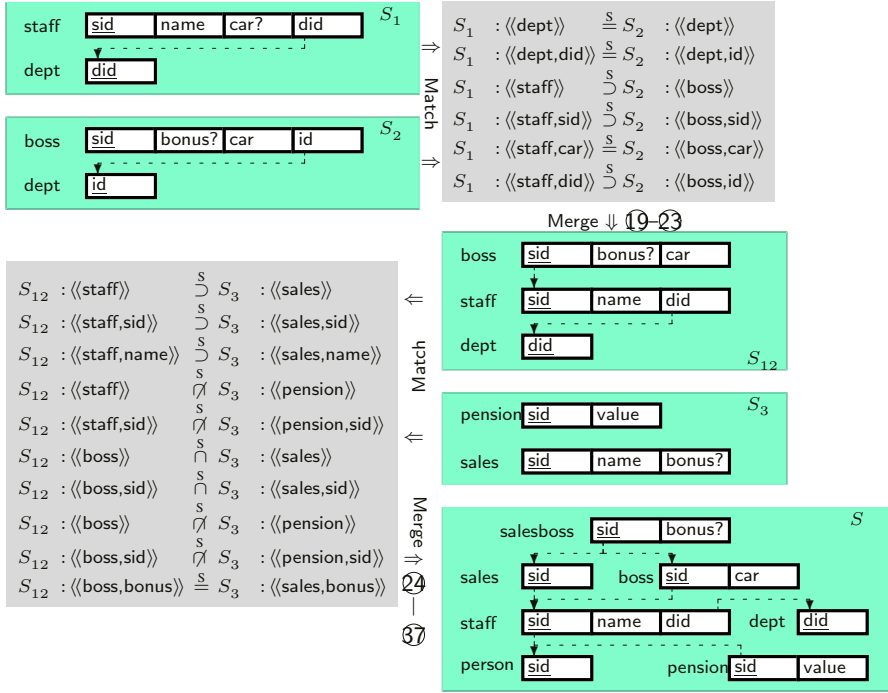
When merging  $S_1^{rel}$  and  $S_2^{rel}$  we have a transformation analogous to ②, except foreign keys are specified on the column of a table rather than the entity class (which ER subsets are defined over):

- ⑳ addFK(⟨⟨⟨boss,sid⟩⟩, ⟨⟨staff,sid⟩⟩⟩)

Then during restructuring, transformation ㉑ is analogous to removing the relationship worksin in ③, but since foreign keys are constraints, no IQL query needs to be supplied. The next two steps are analogous to ④ and ⑤.

- ㉑ deleteFK(⟨⟨⟨staff,did⟩⟩, ⟨⟨dept,did⟩⟩⟩)
- ㉒ deleteColumn(⟨⟨boss,id,nonnull⟩⟩, [{x,y} | {x} ← ⟨⟨boss⟩⟩; {x,y} ← ⟨⟨staff,did⟩⟩])
- ㉓ deleteColumn(⟨⟨staff,car,null⟩⟩, ⟨⟨boss,car⟩⟩)

The resulting relational schema  $S_{12}^{rel}$  is equivalent to the ER schema  $S_{12}^{er}$ . Proceeding to integrate  $S_{12}^{rel}$  with  $S_3^{rel}$ , during merge, the first step is analogous to ⑥. Transformations ㉕–㉘ are similar to ⑦–⑨, but we need to additionally add a ⟨⟨sales,sid⟩⟩ column, since in the relational model each table must have key columns (whereas in the ER model, ⟨⟨salesboss⟩⟩ may inherit the sid attribute from ⟨⟨sales⟩⟩ and ⟨⟨boss⟩⟩). By



**Fig. 2.** Three relational models being integrated. The relational models show the columns of a table in white boxes, with the table name placed to the left of the boxes. Primary key columns are underlined, and nullable columns are followed by a question mark. Foreign keys are shown by drawing dashed arrowed lines

a similar argument (29)–(32) have an extra step compared to (10)–(11). Also, they do not express the semantic constraint that  $\langle\langle \text{pension} \rangle\rangle$  and  $\langle\langle \text{staff} \rangle\rangle$  are disjoint.

- (24) addFK( $\langle\langle \langle\langle \text{sales}, \text{id} \rangle\rangle, \langle\langle \text{staff}, \text{id} \rangle\rangle \rangle\rangle$ )
- (25) addTable( $\langle\langle \text{salesboss} \rangle\rangle, [\{x\} \mid \{x\} \leftarrow \langle\langle \text{sales} \rangle\rangle; \{x\} \leftarrow \langle\langle \text{boss} \rangle\rangle]$ )
- (26) addColumn( $\langle\langle \text{salesboss}, \text{id}, \text{key} \rangle\rangle, [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle \text{sales}, \text{id} \rangle\rangle; \{x, y\} \leftarrow \langle\langle \text{boss}, \text{id} \rangle\rangle]$ )
- (27) addFK( $\langle\langle \langle\langle \text{salesboss}, \text{id} \rangle\rangle, \langle\langle \text{sales}, \text{id} \rangle\rangle \rangle\rangle$ )
- (28) addFK( $\langle\langle \langle\langle \text{salesboss}, \text{id} \rangle\rangle, \langle\langle \text{boss}, \text{id} \rangle\rangle \rangle\rangle$ )
- (29) addTable( $\langle\langle \text{person} \rangle\rangle, \langle\langle \text{pension} \rangle\rangle ++ \langle\langle \text{staff} \rangle\rangle$ )
- (30) addColumn( $\langle\langle \text{person}, \text{id}, \text{key} \rangle\rangle, \langle\langle \text{pension}, \text{id} \rangle\rangle ++ \langle\langle \text{staff}, \text{id} \rangle\rangle$ )
- (31) addFK( $\langle\langle \langle\langle \text{pension}, \text{id} \rangle\rangle, \langle\langle \text{person}, \text{id} \rangle\rangle \rangle\rangle$ )
- (32) addFK( $\langle\langle \langle\langle \text{staff}, \text{id} \rangle\rangle, \langle\langle \text{person}, \text{id} \rangle\rangle \rangle\rangle$ )

During restructuring, transformations (33)–(35) are analogous to (12)–(14). However, when combining  $\langle\langle \text{pension}, \text{id} \rangle\rangle$  and  $\langle\langle \text{staff}, \text{id} \rangle\rangle$  only transformation (36) can be performed — the relational analogy of (15)–(17) — because the columns  $\langle\langle \text{pension}, \text{id} \rangle\rangle$  and  $\langle\langle \text{staff}, \text{id} \rangle\rangle$  are keys and cannot be removed without making the tables invalid. Transformation (37) is analogous to (18). The result of these transformations is  $S_g^{rel}$  in Figure 2.

```

③③ addColumn(⟨⟨salesboss,bonus,null⟩⟩, ⟨⟨sales,bonus⟩⟩)
③④ deleteColumn(⟨⟨sales,bonus,null⟩⟩, ⟨⟨salesboss,bonus⟩⟩)
③⑤ deleteColumn(⟨⟨boss,bonus,null⟩⟩, ⟨⟨salesboss,bonus⟩⟩)
③⑥ addColumn(⟨⟨person,sid,key⟩⟩, ⟨⟨pension,sid⟩⟩ ++ ⟨⟨staff,sid⟩⟩)
③⑦ deleteColumn(⟨⟨sales,name,notnull⟩⟩, [{x, y} |
    {x, y} ← ⟨⟨staff,name⟩⟩; {x} ← ⟨⟨sales⟩⟩])

```

## 4 Representing Models in the HDM

The integration examples in the previous section show that the manual schema transformation and integration processes are driven by intuitive rules based on semantic relationships between schema constructs. The analogy of the rules for the different modelling languages imply that there are also generic rules that hold. In order to capture and define these generic rules a generic framework is necessary, e.g. the **hypergraph data model (HDM)** [23].

A **hypergraph data model (HDM)**  $M$  is a tuple  $\langle Nodes, Edges, Cons \rangle$ , where  $Nodes$  is a set of nodes of a graph,  $Edges$  is a set of nested hyperedges, and  $Cons$  is a set of constraint expressions over the  $Nodes$  and  $Edges$ . In [6] a set of primitive constraint constructs was proposed for the HDM, which will be used here in modelling a higher level modelling language in the HDM:

- **inclusion**  $N_1 \subseteq N_2$ : The extent of node  $N_1$  is a subset of the extent of  $N_2$ .
- **exclusion**  $\mathcal{N}(N_1 \dots N_n)$ : For every  $x, y$  for which  $1 \leq x < y \leq n$ , the extent of node  $N_x$  does not intersect with the extent of  $N_y$ .
- **mandatory**  $N \triangleright E$ : node  $N$  is connected by edge  $E$ , and every instance in the extent of  $N$  must appear at least once in the extent of  $E$ .
- **unique**  $N \triangleleft E$ : node  $N$  is connected by edge  $E$ , and every instance in the extent of  $N$  may appear no more than once in the extent of  $E$ .
- **reflexive**  $N \xrightarrow{id} E$ : when a instance of  $N$  appears in edge  $E$ , then one of the instances of  $E$  is that value of  $N$  as the value of all its nodes. Whilst by itself not very useful, reflexive combined with mandatory and unique defines a notion of a key value.

The HDM model can represent any structured data modelling language [16, 6]. Here we use the approach in [16] that classifies constructs of higher level data modelling language into one of four basic representations in the HDM, which are listed below. Table 1 shows how an illustrative subset of the constructs in Figures 1 and 2 are represented in the HDM.

A **nodal** construct is one that may appear in isolation in a model, and which has an associated extent. For example, an ER model **entity** can be created without being associated to other entities, and represents some set of objects in the UoD. Thus the entity  $\langle\langle staff \rangle\rangle$  is represented in HDM as a single node  $\langle\langle staff \rangle\rangle$ . Since entities have no associated constraints, there are no constraints in the HDM. Relational **tables** are also nodal constructs, are have a very similar mapping to the HDM.

A **link** construct is one that associates other constructs with each other, and which has an extent which is drawn from those constructs. For example, the ER **relationship** construct associates existing entity constructs, and hence is a link construct. Thus,



**Table 1.** Representation of some constructs from  $S_g^{er}$  and  $S_g^{rel}$  in the HDM

node		
sch	construct	scheme
$S_g^{er}$	entity	⟨⟨person⟩⟩
$S_g^{rel}$	table	⟨⟨person⟩⟩
$S_g^{er}$	attribute	⟨⟨person:sid⟩⟩
$S_g^{rel}$	column	⟨⟨salesboss:bonus⟩⟩
$S_g^{er}$	entity	⟨⟨dept⟩⟩
$S_g^{rel}$	table	⟨⟨dept⟩⟩
$S_g^{er}$	entity	⟨⟨pension⟩⟩
$S_g^{er}$	entity	⟨⟨staff⟩⟩
$S_g^{er}$	entity	⟨⟨salesboss⟩⟩
$S_g^{rel}$	table	⟨⟨salesboss⟩⟩

edge		
sch	construct	scheme
$S_g^{er}$	attribute	⟨⟨-,person,person:sid⟩⟩
$S_g^{rel}$	column	⟨⟨-,salesboss,salesboss:bonus⟩⟩
$S_g^{er}$	relationship	⟨⟨worksin,staff,dept⟩⟩

cons			
sch	construct	scheme	op scheme
$S_g^{er}$	relationship	⟨⟨person⟩⟩	▷ ⟨⟨worksin,person,dept⟩⟩
$S_g^{er}$	relationship	⟨⟨person⟩⟩	◁ ⟨⟨worksin,person,dept⟩⟩
$S_g^{rel}$	foreign_key	⟨⟨person⟩⟩	⊆ ⟨⟨person:sid⟩⟩
$S_g^{er}$	attribute	⟨⟨person⟩⟩	$\xrightarrow{id}$ ⟨⟨-,person,person:sid⟩⟩
$S_g^{er}$	attribute	⟨⟨person⟩⟩	◁ ⟨⟨-,person,person:sid⟩⟩
$S_g^{er}$	attribute	⟨⟨person⟩⟩	▷ ⟨⟨-,person,person:sid⟩⟩
$S_g^{er}$	attribute	⟨⟨person:sid⟩⟩	▷ ⟨⟨-,person,person:sid⟩⟩
$S_g^{rel}$	column	⟨⟨salesboss⟩⟩	▷ ⟨⟨-,salesboss,salesboss:bonus⟩⟩
$S_g^{rel}$	column	⟨⟨salesboss:bonus⟩⟩	▷ ⟨⟨-,salesboss,salesboss:bonus⟩⟩
$S_g^{er}$	subset	⟨⟨boss⟩⟩	⊆ ⟨⟨staff⟩⟩
$S_g^{er}$	generalisation	⟨⟨pension⟩⟩	⊆ ⟨⟨person⟩⟩
$S_g^{er}$	generalisation	⟨⟨staff⟩⟩	⊆ ⟨⟨person⟩⟩
$S_g^{er}$	generalisation	⟨⟨pension⟩⟩	⊈ ⟨⟨staff⟩⟩

the ER ⟨⟨worksin,person,dept,1:1,0:N⟩⟩ relationship is represented in the HDM by the edge ⟨⟨worksin,person,dept⟩⟩, which is also associated with constraints that represent the relationship’s cardinality constraints. For example the 1:1 role for ⟨⟨person⟩⟩ in the relationship causes there to be a mandatory and unique constraint in the HDM between HDM nodes ⟨⟨person⟩⟩ and ⟨⟨worksin,person,dept⟩⟩. No constructs in the relational model are link constructs.

A **link-nodal** construct is one that has an associated extent, but may only exist when associated with some other construct. They are represented in the HDM by an edge associating a new node with some existing node or edge. For example, ER **attributes** are link-nodal constructs, and the ⟨⟨person, sid, key⟩⟩ ER attribute is represented by a node ⟨⟨person:sid⟩⟩, and a nameless edge ⟨⟨-,person,person:sid⟩⟩ linking that node to the node representing the entity ⟨⟨person⟩⟩. The fact that an attribute may not exist without its attached entity means that all attributes have a mandatory constraint between the attribute node and the edge (e.g. between ⟨⟨person:sid⟩⟩ and ⟨⟨-,person,person:sid⟩⟩). The key constraint is represented by all mandatory, unique and reflexive constraints

between  $\langle\langle \text{person} \rangle\rangle$  and  $\langle\langle \text{_,person,person:sid} \rangle\rangle$ . If the attribute had been null then the reflexive and mandatory constraints would be omitted, and if the attribute had been notnull then only the reflexive constraint would be omitted. Relational **columns** are also link-nodal constructs, and have a very similar mapping to the HDM as do ER attributes.

A **constraint** construct is one that has no extent associated with it, and just restricts the extent that other constructs may have. For example, the ER **subset** is a constraint construct, the subset  $\langle\langle \text{staff,boss} \rangle\rangle$  is represented by a subset constraint between HDM nodes  $\langle\langle \text{boss} \rangle\rangle$  and  $\langle\langle \text{staff} \rangle\rangle$ . ER **generalisations** are also constraint construct, and are represented by a subset between each child entity and the parent entity, plus an exclusion between the child entities, as illustrated in Table 1 for generalisation  $\langle\langle \text{person,pension,staff} \rangle\rangle$ . Relational **foreign keys** are also constraint constructs, and have a similar mapping to the HDM as do ER subsets.

## 5 Generic Framework for Transformation Generation

Based on the definitions in the previous section, we now define a generic framework for the integration of schemas irrespective of the high level conceptual modelling language used to represent them. We specify a set of **integration rules** that derive BAV transformations from the presence of semantic relationships between nodal, link, and link-nodal HDM constructs. These generic rules can then be translated into high level model specific rules, using techniques from [16, 6]. These higher level model rules are then applied to schemas, and generate BAV transformations such as those presented in Section 3. Four cases of generic rule to specific rule translation are identified:

1. **Exact Translation:** the generic rule can be translated into a model-specific rule by performing a one to one mapping between the HDM constructs and transformations and their model-specific equivalents, *e.g.* an addNode transformation in a generic rule would map into an addEntity transformation in the corresponding ER model rule, and an inclusion constraint would map onto a foreign key constraint in a relational model rule.
2. **Model Limitations:** in some cases the translation of a generic rule in a high level modelling language cannot be exact because a construct or a transformation in the generic rule does not have an equivalent construct or transformation in the high level language. Therefore, some conditions and/or actions of a generic rule might not be translatable. For example, the HDM exclusion constraint cannot be modelled in the relational model, and therefore the addition of such a constraint cannot be translated in a relational model rule.
3. **Meta-constraint Requirements:** because some modelling languages have **meta-constraints**, extra conditions and actions might be necessary for the translation of a generic rule into a model-specific rule. For example, a meta-constraint of the relational model is the existence of a key column for every table. Therefore, a key column must be added by the relational model rules for every table that they add.
4. **Meta-constraint Restrictions:** conditions and/or actions of a generic rule might be restricted in the translated model-specific rule, if they violate the meta-constraints

of the modelling language the rule is translated into, *e.g.* the deletion of a link-nodal construct in a generic rule might be restricted by the corresponding relational model rule, if the link-nodal is a key column.

Since we adopt the standard conform-merge-restructure integration approach [1], integration rules for each stage must be defined. Examples of generic rules for each stage are presented next, together with explanations of their translation into high-level rules for the ER and the relational model, and their application on the schemas in Section 3.

### 5.1 Naming Conforming

In the first stage we deal with naming conflicts: **synonyms** when equivalent constructs have distinct names, and **homonyms** when non-equivalent constructs have identical names. Generic Merge and Distinction rules resolve these two conflicts. Two auxiliary predicates are required at this stage: `identicalNames(C1, C2)` returns *true* when constructs C<sub>1</sub>, C<sub>2</sub> have identical names, *false* otherwise, and `uniqueName(N)` supplies a new name not used by any construct. For example, the Link-Nodal Merge rule:

$$\frac{LN_1 \stackrel{s}{=} LN_2 \quad \neg \text{identicalNames}(LN_1, LN_2)}{\text{renameLN}_{gen}(LN_1, LN_2)}$$

deals with synonymous link-nodals. It examines the existence of the equivalence relationship between two link-nodals LN<sub>1</sub> and LN<sub>2</sub> with non-identical names and assigns to them a common name. The Link-Nodal Distinction rule:

$$\frac{\neg LN_1 \stackrel{s}{=} LN_2 \quad \text{identicalNames}(LN_1, LN_2) \quad \text{uniqueName}(LN')}{\text{renameLN}_{gen}(LN_1, LN')}$$

deals with homonym link-nodals. It assigns to one of them a unique name to explicitly make the two link-nodals distinct. The Nodal and Link Merge and Distinction rules are defined in the same manner.

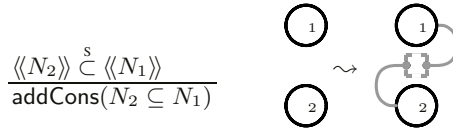
These generic naming conforming rules can be translated into high level models by Exact Translation. Simply, the generic rename transformations will be replaced by the model-specific rename transformations [16]. For example, the Attribute and Column Merge rules for the ER and the relational model are produced from the generic Link-Nodal Merge rule by replacing `renameLNgen` with `renameAttribute` and `renameColumn`, respectively. In the examples of the previous sections, applying these rules would result into transformations ① and ⑱, respectively.

The naming conforming rules satisfy the RPP since the intentional domain of the constructs is not affected, only equivalent constructs are assigned identical names.

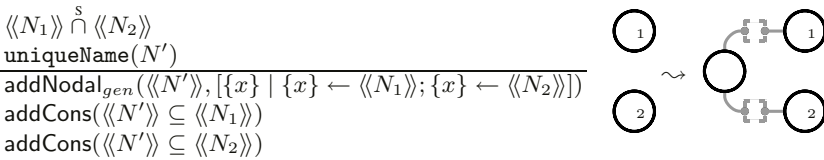
### 5.2 Schema Merging

In the next stage of the integration, the schemas are merged and a single schema is produced. Pair of equivalent constructs, which now have identical names, collapse into

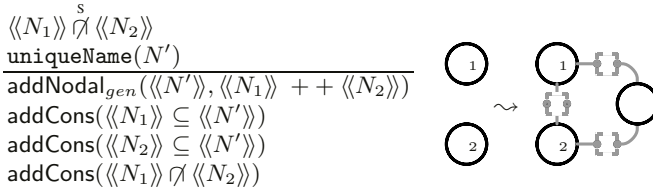
single constructs, new constructs are added and constraints are introduced. The purpose of the rules of this stage is to identify any possible concepts that do not appear explicitly in the schemas. The rules satisfy the RPP since constructs are not deleted from the schema, only added. Therefore the intentional domain of the existing constructs is not affected.



(a) Inclusion Introduction



(b) Addition of Intersection



(c) Addition of Union

**Fig. 3.** Generic Schema Merging Rules

The integration rules at this stage examine the existence of subsumption, intersection and disjointness relationships between nodal constructs. When a subsumption relationship is identified between two nodals then an inclusion constraint must be added between them (Figure 3(a)). When two nodals  $N_1, N_2$  intersect, then a new nodal should be added to represent the common intentional domain of  $N_1$  and  $N_2$ . The appropriate inclusion constraints must also be introduced as illustrated in Figure 3(b). Finally, when two nodals are disjoint, an exclusion constraint is added between them and the union nodal is introduced to represent the union of the disjoint nodal domains (Figure 3(c)).

Exact Translation can be applied on these rules to produce the ER corresponding ones. The  $\text{addNodal}_{gen}$  actions would translate into  $\text{addEntity}$  transformations and the addition of inclusion constraints would become  $\text{addSubset}$  transformations. In Section 3, the ER Inclusion Introduction rule generates transformation ② and the ER Addition of Introduction generates ⑦–⑨ transformations. Finally, the three HDM constraints in the Addition of Union rule map onto an ER generalization, therefore the ER Addition of Union rule can also be produced by Exact Translation. The complete rule, which in our examples generates transformations ⑩–⑪, is defined next:

$$\frac{\langle\langle E_1 \rangle\rangle \overset{s}{\not\sim} \langle\langle E_2 \rangle\rangle}{\text{uniqueName}(E')} \\ \frac{\text{addEntity}(\langle\langle E' \rangle\rangle, \langle\langle E_1 \rangle\rangle + + \langle\langle E_2 \rangle\rangle)}{\text{addGeneralisation}(\langle\langle E', E_1, E_2 \rangle\rangle)}$$

Producing the corresponding merging rules for the relational model does not only require Exact Translation but there is also a Meta-Constraint Requirement and a Model Limitation case. For example, if we examine the Addition of Union rule we have that the generic  $\text{addNodal}_{gen}$  would become an  $\text{addTable}$  transformation by Exact Translation. Because of the Meta-Constraint Requirement of the relational model that each table must have a key column, the rule is required to perform an extra  $\text{addColumn}$  transformation. Conditions  $\text{keyColumn}$  that identify the key columns of the disjoint tables are also additionally added. Notice that the constraints added by the generic rule cannot be represented entirely in the relational model. The Model Limitation is the exclusion constraint, which does not have a corresponding construct in the relational model. Therefore, only the addition of the inclusion constraints is translated (into addition of foreign keys). The complete rule is defined below. An application of it can be seen in transformations 29–32.

$$\frac{\langle\langle T_1 \rangle\rangle \overset{s}{\not\sim} \langle\langle T_2 \rangle\rangle}{\text{uniqueName}(T')} \\ \frac{\text{uniqueName}(KC')}{\text{keyColumn}(\langle\langle T_1 \rangle\rangle, \langle\langle T_1, KC_1 \rangle\rangle)} \\ \frac{\text{keyColumn}(\langle\langle T_2 \rangle\rangle, \langle\langle T_2, KC_2 \rangle\rangle)}{\text{addTable}(\langle\langle T' \rangle\rangle, \langle\langle T_1 \rangle\rangle + + \langle\langle T_2 \rangle\rangle)} \\ \text{addColumn}(\langle\langle T', KC', \text{key} \rangle\rangle, \langle\langle T_1, KC_1 \rangle\rangle + + \langle\langle T_2, KC_2 \rangle\rangle) \\ \text{addFK}(\langle\langle\langle T_1, KC_1 \rangle\rangle, \langle\langle T', KC' \rangle\rangle\rangle) \\ \text{addFK}(\langle\langle\langle T_2, KC_2 \rangle\rangle, \langle\langle T', KC' \rangle\rangle\rangle)$$

### 5.3 Schema Restructuring

In the final stage of the integration, the schema produced during merging is restructured in order to remove structural redundancies. The restructuring rules are defined based on the identified semantic relationships between links and link-nodals. For each relationship between links or link-nodals, all the possible relationships between the corresponding attached nodes are examined. All the possible constraint configurations are also considered. We illustrate this approach with two examples.

Figure 4 examines one case of link subsumption and defines the Generic Optional Link Removal rule. More specifically link  $e_1 = \langle\langle E_1, N_1, N'_{1/2} \rangle\rangle$  subsumes link  $e_2 = \langle\langle E_2, N_2, N'_{1/2} \rangle\rangle$  and node  $\langle\langle N_1 \rangle\rangle$  subsumes  $\langle\langle N_2 \rangle\rangle$ . Since the domain of  $e_2$  is subsumed by  $e_1$ , link  $e_2$  can be considered for deletion. In order to be able to fully restore  $e_2$  after its deletion and hence to satisfy the RPP, it must be ensured that the entities of  $e_1$  that do not appear in  $e_2$  associate with  $\langle\langle N'_{1/2} \rangle\rangle$  only the entities of  $\langle\langle N_1 \rangle\rangle$  that do not appear in  $\langle\langle N_2 \rangle\rangle$ . If this restriction is true then  $e_2$  can be restored by identifying the entities of  $e_1$  that are associated with entities of  $\langle\langle N_2 \rangle\rangle$ . The constraints that force this restriction are:  $\langle\langle N_1 \rangle\rangle \triangleleft \langle\langle E_1, N_1, N'_{1/2} \rangle\rangle$  and  $\langle\langle N_2 \rangle\rangle \triangleright \langle\langle E_2, N_2, N'_{1/2} \rangle\rangle$ . Notice that before the link is deleted any constructs that depend on it have to be examined. Dependent constraints,

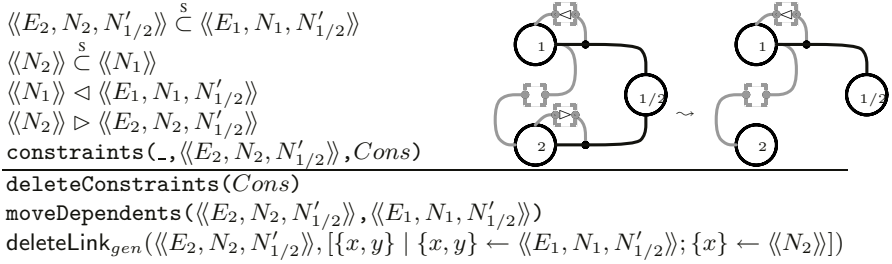
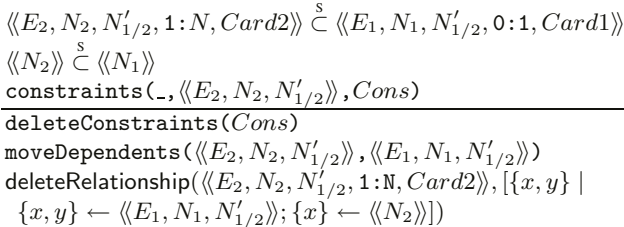


Fig. 4. Generic Optional Link Removal

identified by constraint, are deleted and all other dependent constructs are moved to the remaining link.

The translation of this generic Optional Link Removal rule in the ER language is a simple Exact Translation:



The HDM deleteLink<sub>gen</sub> becomes an deleteRelationship transformation and the mandatory and unique constraints map to cardinality constraints as explained in [6]. The constraints between  $\langle\langle N_1 \rangle\rangle$  and  $e_1$  map into a 0:1 cardinality constraint, which is less restrictive than 1:1, and the mandatory constraint between  $\langle\langle N_2 \rangle\rangle$  and  $e_2$  maps into a 1:N cardinality constraint. In our examples, an application of the ER Optional Link Removal rule generates transformation ③.

Another example of a restructuring rule is illustrated in Figure 5. The case that is examined here is the existence of a disjointness relationship between link-nodal constructs  $\langle\langle X_1, N_1 \rangle\rangle, \langle\langle X_2, N_2 \rangle\rangle$  when  $\langle\langle X_1 \rangle\rangle, \langle\langle X_2 \rangle\rangle$  are also disjoint. In this case, the link-nodal constructs can be generalized by moving them from the sub-nodes  $\langle\langle X_1 \rangle\rangle, \langle\langle X_2 \rangle\rangle$  to the union node  $\langle\langle X' \rangle\rangle$  added during the merging stage and identified by predicate createdNodal. The rule adds the union link-nodal onto  $\langle\langle X' \rangle\rangle$  and then deletes the existing link-nodals. Translating this rule to a high level model (such as transformations ⑮–⑰ in the ER model) requires an examination of Meta-Constraint Restrictions, except from performing Exact Translation of the BAV transformations.

For the ER model, the predicate addLN<sub>gen</sub> can be redefined by Exact Translation. Before performing the corresponding high level transformation, i.e. addAttribute, the common constraints of the existing attributes must be identified and cascaded into the new attribute. Also note that the deleteLN<sub>er</sub> must implement the meta-constraint that either the attribute is not key, or its attached entity is a child of a subset or a generalisation.

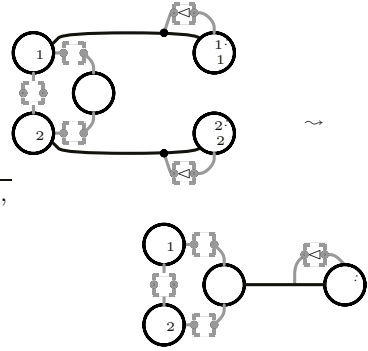
$$\frac{\begin{array}{l} \langle\langle X_1, N_1 \rangle\rangle \overset{s}{\not\sim} \langle\langle X_2, N_2 \rangle\rangle \\ \langle\langle X_1 \rangle\rangle \overset{s}{\not\sim} \langle\langle X_2 \rangle\rangle \\ \text{createdNodal}(X_1, X_2, X') \\ \text{uniqueName}(N') \end{array}}{\begin{array}{l} \text{addLN}_{gen}(\langle\langle X', N' \rangle\rangle, [\langle\langle X_1, N_1 \rangle\rangle] ++ [\langle\langle X_2, N_2 \rangle\rangle], \\ \langle\langle X_1, N_1 \rangle\rangle, X_2, N_2) \\ \text{deleteLN}_{gen}(\langle\langle X_1, N_1 \rangle\rangle, [\{x, y\} | \\ \{x, y\} \leftarrow \langle\langle X', N' \rangle\rangle; \{x\} \leftarrow \langle\langle X_1 \rangle\rangle]) \\ \text{deleteLN}_{gen}(\langle\langle X_2, N_2 \rangle\rangle, [\{x, y\} | \\ \{x, y\} \leftarrow \langle\langle X', N' \rangle\rangle; \{x\} \leftarrow \langle\langle X_2 \rangle\rangle]) \end{array}}$$


Fig. 5. Link-Nodal Generalisation

$$\text{addLN}_{er}(\langle\langle X, N \rangle\rangle, Q, \langle\langle X_1, N_1, C_1 \rangle\rangle, \langle\langle X_2, N_2, C_2 \rangle\rangle) :- \\ C_1=C_2, \text{addAttribute}(\langle\langle X, N, C_1 \rangle\rangle, Q).$$

$$\text{deleteLN}_{er}(\langle\langle X, N, C \rangle\rangle, Q) :- \\ \neg C = \text{key}, \langle\langle X', X \rangle\rangle, \langle\langle X', \dots, X, \dots \rangle\rangle, \text{deleteAttribute}(\langle\langle X, N, C \rangle\rangle, Q).$$

Translating the rule in the relational modelling language, an extra restriction is required. In the redefinition of  $\text{addLN}_{gen}$  a new column cannot be added if it is the union of key columns, because table  $\langle\langle N' \rangle\rangle$  has already got a key column, added by the Addition of Union rule. In the case of  $\text{deleteLN}_{rel}$ , a Meta-Constraint Restriction applies which does not allow the deletion of key columns.

$$\text{addLN}_{rel}(\langle\langle X, N \rangle\rangle, Q, \langle\langle X_1, N_1, C_1 \rangle\rangle, \langle\langle X_2, N_2, C_2 \rangle\rangle) :- \\ C_1=C_2, \neg C_1=\text{key}, \text{addColumn}(\langle\langle X, N, C_1 \rangle\rangle, Q). \\ \text{deleteLN}_{rel}(\langle\langle X, N, C \rangle\rangle, Q) :- \\ \neg C = \text{key}, \text{deleteColumn}(\langle\langle X, N, C \rangle\rangle, Q).$$

## 6 Related Work

Many approaches to generating schema transformations can be found in the literature. Early work can be found in [12, 21], where formal definitions of semantic relationships between schema constructs similar to ours are given. However, both approaches are concerned with integrating schemas defined in an extended ER language, which induces restrictions compared to our generic approach of using the low-level HDM. We define a wider set of formal rules and examine all possible constraint configurations.

In [10] similar semantic relationships to ours are used, where schema integration is performed based on corresponding ontologies and concepts. However, the steps for the creation of the integrated schema are not formally defined, nor is the data mapping, and further restrictions are imposed, *e.g.* one schema construct can only map to only one other construct.

The work most related to ours is [3], where a low-level graph-based modelling language is also adopted, called Vanilla, which models both the schemas and the correspondences between their constructs. There is an example showing how an extended ER language can be supported by Vanilla, however there is no extensive explanation of how schemas can be translated from Vanilla into a high level modelling language.

The advantage of using the HDM as the common modelling language is that the translation to and from Relational, ER, XML and UML schemas [16, 17] has already been studied. Additionally, the schema integration approach in [3] is based only on semantic equivalence between nodes, while we deal with a wider range of semantic relationships between all types of generic constructs (nodals, links and link-nodals). However, the advantage of [3] is that data-level correspondences between constructs are also considered, *e.g.* data level correspondence would specify that the instances of two constructs can be concatenated. Our approach has a more semantic perspective than a data-level one. Another difference between the two approaches is that we explicitly deal with constraints and they are a necessary part of our rules. Finally, as for [19] where another schema integration approach is proposed based entirely on equivalence relationships, our methodology has the advantage of not only removing integrating schemas but additionally removing structural redundancies.

## 7 Summary and Conclusions

In this paper, we have presented a generic and formal framework to generate schema transformations in the Merge operator. We use the low level HDM as the common data modelling language, which permits the extension of this framework to any higher-level modelling language. Our integration rules take as input four types of semantic relationships — equality, subsumption, disjointness and intersection — and generate BAV transformations over the HDM. Using the correspondence between the HDM and higher-level models, these rules can be translated into rules that apply to higher-level models. In this paper examples of translating generic rules into ER and relational modelling language rules have been presented.

Since we adopt the BAV integration methodology, we are able to reason about the transformation steps, demonstrate that we preserve information during the integration and prove the correctness of the process.

Since we deal with a wide variety of semantic transformations, our framework can be used in conjunction with most schema matching techniques [14, 15] both for merging and improving schema structure.

In future work we will consider more complicated mappings as in [8, 20, 9], and rules for removing redundant constraints. Relationships between different types of constructs might also prove useful. Our target is to implement a tool that based on this formal framework can assist in the automatic integration of schemas.

## References

1. C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
2. P. Bernstein. Applying model management to classical meta data problems. In *Proc. CIDR, 2003*, 2003.
3. Philip A. Bernstein and Rachel A. Pottinger. Merging models based on given correspondences. In *Proc. 29th VLDB Conference*, Berlin, 2003.



4. M. Boyd, S. Kittivoravittkul, C. Lazanitis, P.J. McBrien, and N. Rizopoulos. AutoMed: A BAV data integration system for heterogeneous data sources. In *Proc. CAiSE2004*, volume 3084 of *LNCS*, pages 82–97. Springer-Verlag, 2004.
5. M. Boyd, S. Kittivoravittkul, C. Lazanitis, P.J. McBrien, and N. Rizopoulos. Overview of the automated repository. Technical Report No. 26, AutoMed, 2004.
6. M. Boyd and P.J. McBrien. Towards a semi-automated approach to intermodel transformations. In *Proc. EMMSAD 04, CAiSE Workshop Proceedings Volume 1*, pages 175–188, 2004.
7. P. Buneman et al. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, 1994.
8. Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Halevy, and Pedro Domingos. iMAP: discovering complex semantic matches between database schemas. In *Proc. SIGMOD 2004*, pages 383–394. ACM Press, 2004.
9. Anca Dobre, Farshad Hakimpour, and Klaus R. Dittrich. Operators and classification for data mapping in semantic integration. In *Proc. ER 2003*, pages 534–547, 2003.
10. F. Hakimpour and A. Geppert. Global schema generation using formal ontologies. In *Proc. ER02*, volume 2503 of *LNCS*, pages 307–321. Springer-Verlag, 2002.
11. E. Jasper, N. Tong, P.J. McBrien, and A. Poulouvasilis. View generation and optimisation in the AutoMed data integration framework. In *Proc. Baltic DB&IS04*, volume 672 of *Scientific Papers*, pages 13–30. Univ. Latvia, 2004.
12. J.A. Larson, S.B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, April 1989.
13. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS'02*, pages 233–246. ACM, 2002.
14. L.Xu and D.W. Embley. Discovering direct and indirect matches for schema elements. In *8th International Conference on Database Systems for Advanced Applications (DASFAA '03), Kyoto, Japan, March 26–28, 2003*, pages 39–46, 2003.
15. J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proc. 27th VLDB Conference*, pages 49–58, 2001.
16. P.J. McBrien and A. Poulouvasilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99*, volume 1626 of *LNCS*, pages 333–348. Springer, 1999.
17. P.J. McBrien and A. Poulouvasilis. A semantic approach to integrating XML and structured data sources. In *Proc. CAiSE'01*, volume 2068 of *LNCS*, pages 330–345. Springer, 2001.
18. P.J. McBrien and A. Poulouvasilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, pages 227–238. IEEE, 2003.
19. Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: a programming platform for generic model management. In *Proc. SIGMOD 2003*, pages 193–204. ACM Press, 2003.
20. R.J. Miller, M.A. Hernández, L.M. Haas, L.-L. Yan, C.T.H. Ho, R. Fagin, and L. Popa. The Clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.
21. C. Parent and S. Scappapoetra. View integration: A step forward in solving structural conflicts. Research Report, EPFL-Computer Sc. Dept. Lausanne, 1990.
22. A. Poulouvasilis and M. Levene. A nested-graph model for the representation and manipulation of complex objects. *ACM Trans. on Information Systems*, 12(1):35–68, 1994.
23. A. Poulouvasilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
24. S. Scappapietra and C. Parent. View integration: A step forward in solving structural conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):258–274, April 1994.
25. C. Zaniolo and M. Melkanoff. A formal approach to the definition and the design of conceptual schemata for database systems. *ACM TODS*, 1982.

# Building a Software Factory for Pervasive Systems Development\*

Javier Muñoz and Vicente Pelechano

Departamento de Sistemas Informáticos y Computación,  
Universidad Politécnica de Valencia,  
Camí de Vera s/n, E-46022, Spain  
{jmunoz, pele}@dsic.upv.es

**Abstract.** The rise of the number and complexity of pervasive systems is a fact. Pervasive systems developers need advanced development methods in order to build better systems in an easy way. Software Factories and the Model Driven Architecture (MDA) are two important trends in the software engineering field. This paper applies the guidelines and strategies described by these proposals in order to build a methodological approach for pervasive systems development. Software Factories are based on the definition of software families supported by frameworks. Individual systems requirements are specified by means of domain specific languages. Following this strategy, our approach defines a framework and a domain specific language for pervasive systems. We use the MDA guidelines to support the development of our domain specific language and the automatic generation of the specific source code of a particular system. The approach presented in this paper raises the abstraction level in the development of pervasive systems and provides high reusable assets to reduce the effort in the development projects.

## 1 Introduction

Computing based systems growth is arriving to all environments of our daily life. Pervasive systems live around us providing services to the inhabitants of a home, the workers of an office or the drivers in a car park. We know that requirements for current and future pervasive systems involve a great diversity of types of services [14]. Such different services as multimedia, communication or automation services need hardware devices that different manufacturers provide. These devices live in several networks running on different platforms. The development of such systems is a very hard task because it should achieve devices interoperability in an heterogeneous environment in order to satisfy system requirements.

Therefore, there is a need of new solid engineering methods for developing robust pervasive systems. Recently, two compatible approaches have been proposed

---

\* This work has been developed with the support of MEC under the project DESTINO TIN2004-03534 and cofinanced by FEDER.

for developing software systems in a highly productive and cost-effective way. Software Factories [4] and the Model Driven Architecture (MDA) [10] provide strategies for raising the abstraction level in the software development process and making affordable the development of complex systems. The application of the guidelines defined in this approaches to pervasive systems development can help to build better systems in an easier way than applying traditional methods. Software Factories focus on producing reusable assets that reduce the overall development time. On the other hand, MDA promotes the use of high abstraction level models which provide the system developers with an intuitive way for describing the system. These models should be automatically transformed to the final implementation code.

The work presented in this paper proposes a methodological approach to pervasive systems development following Software Factories principles and MDA guidelines. The contribution of is this work is double. On the one hand, we apply the Software Factories proposal to a concrete domain. Then, this work provides an application case that can be used as a recipe for the construction of Software Factories for other domains. On the other hand, our approach contributes to the state of the art in pervasive systems development. We provide a model driven development method for the specification and implementation of pervasive systems. Our approach establishes a methodological framework for automating the construction of high-quality pervasive systems in a productive way

The structure of the paper is the following: Section 2 briefly introduces the Software Factories and MDA approaches. We justify the application of these proposals to Pervasive Systems development. Section 3 describes Pervasive Systems main characteristics and it presents our point of view for developing these kind of systems. We introduce a Pervasive System for a meetings room in order to illustrate our approach. Section 4 describes our application of Software Factories and MDA to Pervasive Systems. We present our strategy to apply MDA guidelines into our methodological approach. We propose techniques for every MDA building block. The Pervasive Modeling Language (Perv-ML), a modelling language for describing Pervasive System using high abstraction level constructs, is introduced. Next, a framework for developing Pervasive Systems is presented. Finally, section 5 includes some conclusions and further work.

## 2 Software Factories and MDA

A **Software Factory**, as defined in [4], is a software development organization that produces similar systems encouraging the reuse of architectures, components and know-how. Therefore, Software Factories focus on the development of similar systems encouraging the reuse of architectures, components and know-how.

On the other hand, MDA is, as described in the IEEE Software special issue on Model Driven Development [8], a software development approach that uses models to drive the development of software systems. MDA is a software development approach that uses models to drive the development of software systems.

Following this approach, system developers build high level abstraction models (called Platform Independent Models, PIM) and transform them obtaining models that directly represent the final software product (called Platform Specific Model, PSM).

Therefore, there is a natural integration of these two approaches. MDA techniques can be used to support the development of domain specific languages for building high level abstraction models. Then, these models can be transformed in order to obtain the specific source code of a system in the context of a family of systems.

Although the use of OMG languages is explicitly avoided and criticized in [4], we think that the use of standards is a key aspect for the success of these approaches, even when the standards are not as good as we would like.

In short, we are interested in the **strengths of both approaches**:

- ... we get their focus on reuse by means of domain specific development.
- ... we get their focus on high level abstraction models and automatic code generation.

In order to describe this specific domain, next section points out pervasive systems main characteristics.

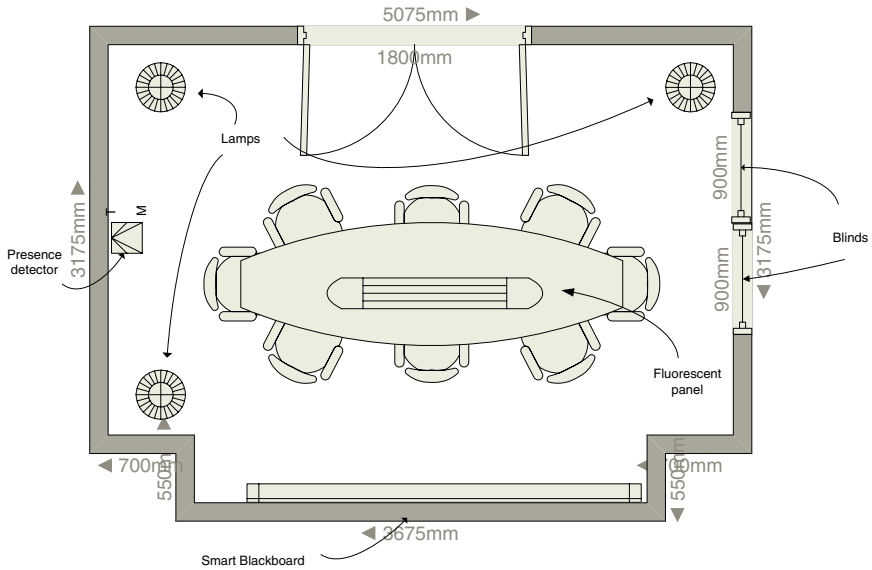
### 3 Pervasive Systems

Pervasive systems try to build environments where computation elements disappear from the user point of view but their functionality is still provided. This vision was initially described by Weiser [15] in the early 90s and it is based on the construction of computing-saturated environments properly integrated with human users. The big challenge of this vision is the integration of several existing technologies (handheld computers, broadband communications, sensor devices, etc.) in an homogeneous whole. The development of such a kind of systems requires the contribution of several engineering and research fields: hardware designers, human-computer interaction experts, software engineers, etc. We can find pervasive systems in environments like cars, offices, public building and, of course, our homes.

Requirements for current and future pervasive systems involve a great diversity of types of services [14]. Such different services as multimedia, communication or automation services need hardware devices that different manufacturers provide and external software systems. These elements live in several networks running on different technological platforms, but they can not satisfy isolatedly all system requirements. The elements that compose the system must work together for achieving some system goals. Therefore we can distinguish two sources of service providers: **commercial off-the-shelf (COTS) elements**<sup>1</sup> and the **software system** that integrates all the elements of the pervasive system.

---

<sup>1</sup> We extend the definition of COTS to include hardware devices



**Fig. 1.** The map of a smart meetings room

Considering this point of view, the development of a pervasive system consists of :

- **The selection of the suitable COTS devices or external software systems.** These elements should provide the services that users require either isolatedly or interacting with other elements.
- **The development of the software system that integrates the external elements in order to provide the services that users require.** The development of that software may imply the use of different technologies but some gateway technology should exist.

We describe a pervasive system for a meetings room in order to illustrate our approach. In such a system, depicted in Fig. 1, users require services like *presence detection* by *presence detector* or *illumination*. Users do not mind what devices compose the system, they just need a specific functionality. System architects deal with selecting the most suitable devices (like *presence detector* or a *presence detector* in our case of study) for providing that functionality.

## 4 A Software Factory for Pervasive Systems Development

As outlined in section 3, the development of a pervasive system implies the use of many different technologies in order to satisfy all users requirements [13, 6]. Usually these technologies provide low abstraction level constructs to the developer.

Therefore, applying a MDA approach to pervasive systems supposes jumping a very wide abstraction gap that must deal with the technology heterogeneity.

Following the Software Factories approach, a framework for pervasive systems should be developed applying domain engineering principles. This framework raises the abstraction level of the target platform and, therefore, the amount of code is sensibly reduced.

Thus our proposed methodological approach to pervasive systems development is based on:

- the construction of a domain specific language for the description of pervasive systems.
- the construction of a framework that raises the abstraction level by providing similar constructs to those defined by the domain specific language.
- the definition of mappings or rules for the transformation of models, that are built using the domain specific language, to code that fulfils the defined framework.

Next subsections describe both the MDA point of view of the proposed approach and the main architecture of the framework for pervasive systems. Subsection 4.1 presents the techniques for the construction of the domain specific language and the definition of the mapping rules. Subsection 4.2 introduces the implementation framework for developing pervasive systems.

### 4.1 MDA for Pervasive Systems

As we have justified in section 2, the MDA approach can be used in a Software Factory to support the development of domain specific languages and the automatic code generation step. The standard defines several building blocks for the definition of MDA based methods, but it does not specify concrete techniques to be used in each step. In order to put MDA in practice we should provide concrete techniques for each building block. These techniques must be defined using OMG standards. Our approach proposes the following techniques for applying MDA (see Fig. 2) to pervasive systems development:

1. **A precise language for building Platform Independent Models (PIMs).** This is the domain specific language for precisely describing pervasive systems using high abstraction level constructs. We have defined the Pervasive Modeling Language (Perv-ML) (outlined next) in order to build PIMs.

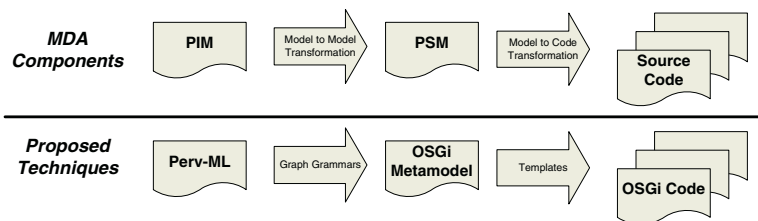


Fig. 2. MDA building blocks and our proposed techniques for pervasive computing

2. One or many **modelling languages for building Platform Specific Models** (PSMs). The conceptual primitives of these languages must be direct representations of constructs of the technology they model. In our case, the target platform is the framework for pervasive systems developed as result of the domain engineering activity. As we will see next, this framework is tightly based on OSGi [7]. OSGi is a standard defined by the Open Service Gateway Initiative (OSGi) that describes a framework that was initially created for hosting software of residential gateways. Then, we have defined an OSGi metamodel for building Platform Specific Models.
3. **PIM to PSM transformations**. These transformations define how a PIM can be converted to a PSM. Currently, model transformations is a hot research topic. We apply graph grammars for defining the transformations from Perv-ML to OSGi.
4. **PSM to source code transformations**. Finally, the code generation from the PSMs is the last step of the development method. We are applying templates to the elements of models in order to obtain the source code.

#### 4.1.1 Pervasive Modelling Language (Perv-ML): The PIM Language

Perv-ML is a language designed with the aim of providing the system analyst with a set of constructs that allow to precisely describe the pervasive system. Perv-ML promotes the separation of roles where developers can be categorized as analysts and architects. Fig. 3 shows the language organization. The dashed arrow of Fig. 3 defines the construction order of the conceptual models that our approach proposes. In short, systems analysts capture system requirements and describe the pervasive system at a high level of abstraction using the service metaphor as the main conceptual primitive. Analysts build three graphical models that constitute what we call the **Analyst View**. On the other hand, system architects specify what COTS devices and/or existing software systems realize system services. Architects build other three models that constitute what we call the **Architect View**. Next we give a more detailed description of the language.

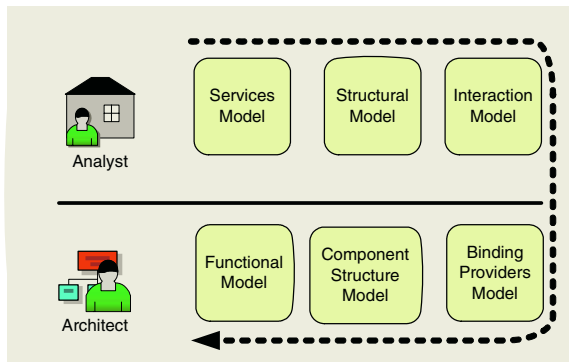


Fig. 3. The six models of Perv-ML

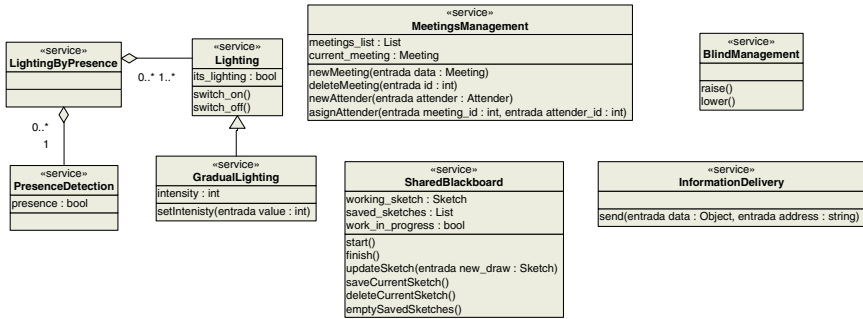


Fig. 4. Meetings room Services Model

**The Analyst View.** The Analyst describes a pervasive system specifying a set of functional elements that provide a specific set of services that the user of the system requires. Those functional elements are what we call service instances. For instance, if the meeting room described above has two binds and any user wants to control them independently, the pervasive system must provide two elements (instances) that provide the bind management service. Following this approach we propose a step previous to the building of the Pervasive System Conceptual Structure. In this first step, we introduce the **Services Model** where the analyst defines services and their relationships. Perv-ML uses and extends UML Class Diagram for representing the description of the services, and the State Transition Diagram for modelling the behaviour. Fig. 4 shows the Service Model of our meeting room.

Analyst defines the pervasive system functional structure in the **Structural Model**. This model specifies the service instances of the system which are represented by a component. Perv-ML provides components as abstractions of the low-level elements that realize the services. Every system component provides one of the services described in the Services Model. In Fig. 5. we can see that the **LightingManagement** component has dependency relationships with the **MainLighting** and the **Presence** components due to the aggregation relation-

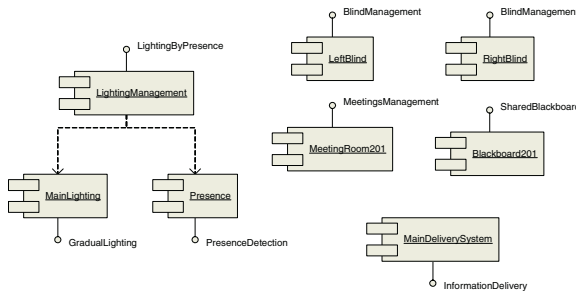
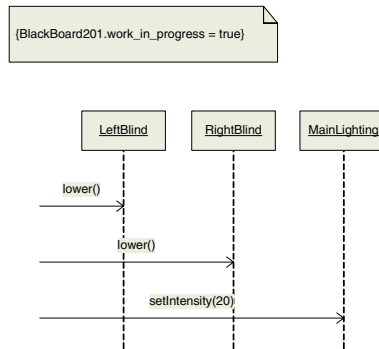


Fig. 5. Components that provide the services of our case of study system





**Fig. 6.** An interaction that lowers blinds and sets lighting to 20% of its maximum intensity

ship defined in the Services Model. Perv-ML represents the Structural Model as a UML Component Diagram.

As we have said in section 3, system services must cooperate in order to satisfy all the system requirements. Analyst describes services cooperation in the **Interaction Model**. An interaction is a communication between services for providing a specific functionality, so analyst must describe as many interactions as joint functionality the system provides. Every interaction is described by an adapted UML Sequence Diagram, therefore the Interaction Model is composed by several sequence diagrams. Fig. 6 shows an interaction for suiting lighting when the blackboard service is being used. It lowers both blinds and it sets the lighting service at a 20% of its maximum power. This interaction takes place when somebody starts using the blackboard.

**The Architect View.** We need to build a detailed specification of the lower level artefacts that realize system services in order to have a complete and operative pervasive system description. We use the term *binding providers* for referring artefacts that the pervasive system manages to interact with its physical or logical environment. A *device*, a *component*, an *actuator* or an *entity* can be binding providers. Architect describes every binding provider type that is introduced to implement system services in the **Binding Providers Model**. A type of binding provider represents a set of devices or software systems that provide a similar functionality without detailing manufacturer specific information. The Binding Provider Model is depicted using a stereotyped UML Class Diagram. Fig. 7 shows some binding providers of our meeting room. The usage of **Lamp** and **FluorescentPanel** actuators is different although both can be used for lighting a room.

The System architect uses the **Component Structure Specification** to specify the bindings providers that realize a component of the Structural Model. For instance, a component that provides a lighting management service can be realized by three lamps and a fluorescent panel. In such a case, the Binding Providers Model must contain the lamp description. See Fig. 8 of the Structure

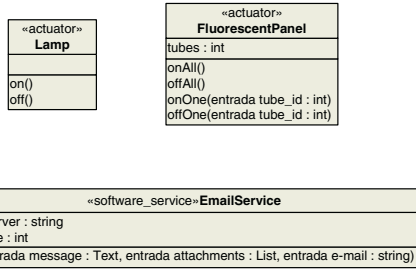


Fig. 7. Some elements of a Binding Providers Model

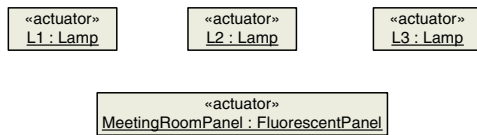


Fig. 8. Structure Specification of the MainLighting component

Specification for the MainLighting component included in our meeting room Structural Model (see Fig. 5).

Finally, architect must specify how every component operation is realized. In the **Component Functional Specification** architect defines the sequence of actions that the component realize when an operation is invoked. Architect specifies actions using the UML Action Semantic Language (ASL). ASL does not have an official concrete syntax, but many proposed syntaxes are available like the one by Kennedy Carter [16].

Using the Perv-ML approach the system is completely described in a technology and manufacturer independent way. When a new technology emerges, system description does not need to be modified. Moreover, if we want to use a device of a new manufacturer we only have to develop a driver that adapts its interface to the generic interface used in the Binding Providers Model. Even if the system architect decides to change a component specification, analyst view remains unmodified. We have isolated changes by means of stratification through abstraction levels.

#### 4.1.2 OSGi Metamodel: The PSM Language

As described above, there are a lot of implementation technologies for developing pervasive systems. Using only a low-level technology for control (LonWorks, EIB, UPnP) , data (Ethernet, Bluetooth, WiFi) or multimedia (IEEE1394, HAVi) networks is not possible because of the diversity of services required, therefore we have selected OSGi, a middleware platform that has bridges many of them and provides high-level constructs for building pervasive systems. This

middleware help us notably for filling the abstraction gap between the domain specific language and the target implementation technology.

The Open Service Gateway Initiative (OSGi) [7] is an association of companies, that includes Sun Microsystems, IBM, Oracle and Nokia, created with the aim of developing an open standard for service gateways. A service gateway is the platform where resides the software for providing home services. It manages home devices and it communicates with external networks. The standard defines Java APIs for libraries that the OSGi platform provides and several standard services like Logging, HTTP Server, Device Management, etc. Our own framework is built on top of this middleware using their runtime environment and services.

In order to integrate OSGi in the MDA phase of our development method, we have to create models which are built using OSGi concepts. We have developed an OSGi metamodel for specifying these concepts and their relationships. The models built with a OSGi-only metamodel cannot be seen as final implementation models because every OSGi concept is actually implemented as a Java entity. For instance, an OSGi Bundle is implemented as a JAR package, and an

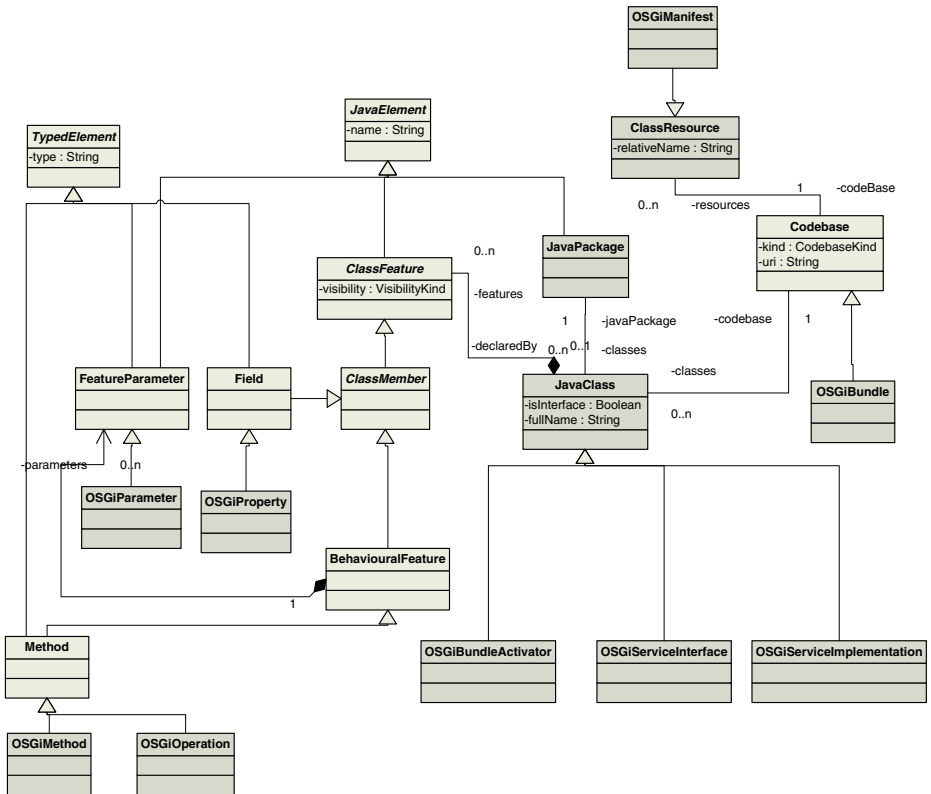


Fig. 9. An OSGi/Java metamodel

OSGi Service Implementation is implemented through a Java Class. Therefore, a fully functional modelling language for specifying OSGi based systems should include a complete Java metamodel. Then, the inclusion of OSGi concepts can be done as Java entities extensions with new specific constraints.

Our complete OSGi/Java metamodel is based on the Java metamodel developed by the NetBeans Community <sup>2</sup>. This metamodel has been adapted and extended to fit it in our needs. Fig. 9 shows a view of the Java metamodel with our extensions. Elements that are mapped from the OSGi metamodel have been depicted in grey.

### 4.1.3 Graph Grammars. Defining the Model Transformation Engine

As noted earlier, the definition of transformations between PIM and PSM involve jumping a wide gap between abstraction levels. Currently standards for the definition of transformations do not exist [2]. OMG published a *Model Transformation Language* [9] in order to achieve a language for defining transformation between metamodels built with its Meta Object Facility (MOF). In the meantime, we are using graph grammars [3] as the model transformation engine. There exist many works [1, 5, 12] that propose graph grammars as a suitable technique for model transformation. From a mathematical point of view, a model can be seen as a graph where model elements are labelled nodes and the relationships between model elements are edges. In this way we can apply all the existing knowledge for defining graph transformations in order to achieve model transformations in the MDA context. Graph grammars have many advantages over other proposed techniques: a formal mathematical sound, algorithms for their application and a graphical representation for intuitively defining transformations.

Fig. 10 shows two rules for model transformation from Perv-ML models to OSGi-based models. Every rule is composed by a Left Hand Side (LHS), that defines a pattern to be matched in the source graph, and a Right Hand Side that defines the replacement for the matched subgraph. For instance, first rule says that when a Perv-ML **Component** element is found it must be transformed into a **Bundle** element and references to a **Java Class** and **Manifest** elements have to be created and linked to the **Bundle**. Following this approach, transformation from Perv-ML models to OSGi-based models is defined following a set of rules like those defined in this section.

## 4.2 A Framework for Pervasive Systems Development

The framework for pervasive systems has been developed for supporting Perv-ML, the domain specific language for this kind of systems. Therefore, the scope and approach of the framework is inherited from the language. This means that, as in Perv-ML, the framework is based on the assumption that the software of a pervasive system must integrate many devices and external software systems in order to provide the services that the users require. Following this approach, users should deal with services and the software is in charge of the management of

<sup>2</sup> <http://java.netbeans.org/models/java/java-model.html>

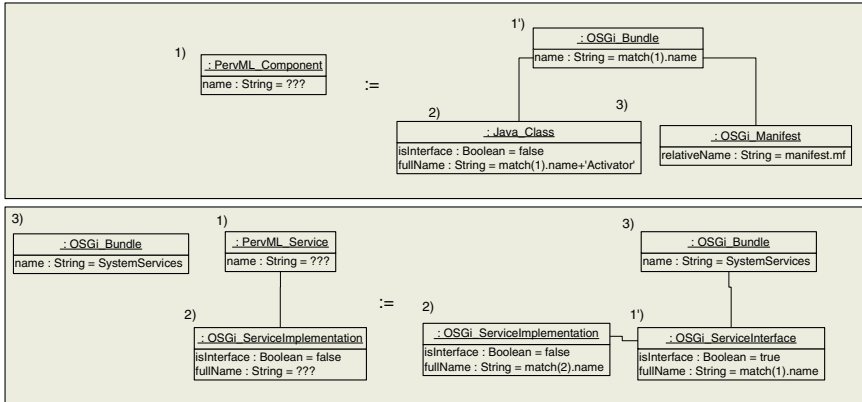


Fig. 10. Two rules that define models transformation

the devices or external systems for providing that services. Then, our framework provides implementation primitives for directly supporting Perv-ML conceptual primitives.

As described in subsection 4.1, we are using OSGi as platform for the development of pervasive systems. This technology fits smoothly in our approach and minimizes the abstraction gap to be filled. Many Perv-ML conceptual primitives maps directly to OSGi implementation concepts, so OSGi can be considered a key component of our framework.

Several architectures can be used when developing with OSGi. Fig. 11 shows the global structure of the systems developed with our method. Packages in the figure represent sets of resources (classes, interfaces, icons, etc.) with a common goal. Dashed arrows represent dependence relationships. For instance, the `.-` package requires some resource located in the `.-` package. We use a three-tier architecture adapted to this kind of systems. Layers of the architecture are described next.

### 4.2.1 User Interface Layer

The user interface layer is currently implemented as web pages using the HTTP Service integrated in the OSGi platform. We divide this layer in two components.

- The **main user interface** is the entry door to the system and is in charge of the organization of the access to the system services (by localization, by kind of service, by more used services, etc.) and security issues.
- The **individual services interface** manage the interaction of every particular service in the system. Services of the same type have the same user interface. For instance, in Fig. 11 services 2 and 3 (maybe lighting services) are managed by the user interface 2 (UI2).

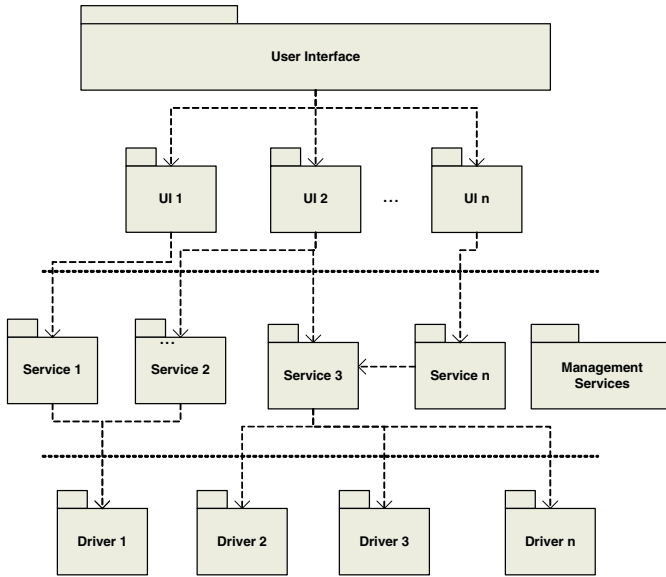


Fig. 11. Global architecture of the framework

### 4.2.2 Logical Layer

Elements in the logical layer can be classified in two groups attending their purpose:

- **Services for supporting the functionality specified in the Perv-ML model.** These services are implemented as Java Classes and registered as OSGi Services. All of them must implement the `Service` interface for ensuring a proper execution of the system. This interface has operations for checking invariant constraints, managing concurrent execution, error handling, etc.
- **Services for the management of the system execution.** This set of services are in charge of ensuring global constraints satisfaction, checking trigger conditions, providing web services and other auxiliar functionality. These services are common to all the applications based on this framework.

### 4.2.3 Communication Layer

Finally, the **Communication Layer** manages the interaction of the pervasive system with its physical or logical environment. This layer is composed by drivers that are used by the services in the upper layer. Every device or external software system is represented in this layer by a driver. Drivers in this layer are implemented as Java Classes and registered as OSGi Services.

For avoiding interface heterogeneity, a unified interface is used for all similar devices (or software systems). This means that, for instance, in our example

there is a unique interface for all lamps, all instant messaging systems or all video projectors. Then, the driver is in charge of adapting that interface to the actual device interface. The selection of the specific drivers for every generic device interface happens in model compilation time.

## 5 Conclusions

In this paper we have presented a methodological approach for the development of pervasive systems. Following the Software Factories strategy, our approach is based on the construction of a framework for a family of similar systems and a domain specific language (Perv-ML) for the specification of family members requirements. We follow the MDA standard for the definition of the domain specific language and the automatic code generation step. This merged approach can help to build better pervasive systems in an easier way than applying traditional methods.

We have experimented many of these benefits in the development of Information Systems. Our research group have developed a model driven method (called OO-Method [11]) with full code generation capabilities that has been implemented in the OlivaNova Model Execution System<sup>3</sup>. Our aim is to apply these successful ideas to pervasive systems development. This work was initially developed in the context of a R&D project together with Telefonica I+D. Perv-ML has been applied for the specification of applications for Smart Homes.

We are currently working on providing tool support for several steps of the method, like the construction of models using Perv-ML and the automatic transformations of that models. Another important ongoing work is the specification of the transformation rules from Perv-ML to OSGi code to implement the specific part of the framework. Finally, we are implementing pervasive systems with our framework in order to obtain feedback for tuning our proposal.

## References

1. György Csertán, Gábor Huszerl, István Majzik, Zsigmond Pap, András Pataricza, and Dániel Varró. VIATRA: Visual automated transformations for formal verification and validation of UML models. In Julian Richardson, Wolfgang Emmerich, and Dave Wile, editors, *Proc. ASE 2002: 17th IEEE International Conference on Automated Software Engineering*, pages 267–270, Edinburgh, UK, September 23–27 2002. IEEE Press.
2. Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 2003.
3. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook on Graph Grammars and Computing by Graph Transformation*, volume 2 Applications, Languages and Tools. World Scientific Publishing Co., Inc., 1999.

---

<sup>3</sup> <http://www.care-t.com/>

4. Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories*. Wiley Publishing Inc., 2004.
5. Reiko Heckel, Jochen Küster, and Gabriele Taentzer. Towards automatic translation of UML models into semantic domains. In *Proc. AGT 2002: Workshop on Applied Graph Transformation*, pages 174–188, 2002.
6. M. Hwang, Y. Jeon, and J. Kim. Standarization activities and technology competitors for the in-home networking. In *Internation Conference on Communication Technology*, 1998.
7. Dave Marples and Peter Kriens. The Open Services Gateway Initiative: An Introductory Overview. *IEEE Communications Magazine*, 39(12):110–114, 2001.
8. Stephen J. Mellor, Anthony N. Clark, and Takao Futagami. Guest editors' introduction: Model-driven development. *IEEE Software*, 20(5):14–18, 2003.
9. Object Management Group. OMG MOF 2.0 Query, Views, Transformations Request for Proposals.
10. Object Management Group. Model Driven Architecture Guide, 2003.
11. Oscar Pastor, Jaime Gómez, Emilio Insfrán, and Vicente Pelechano. The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. *Information Systems*, 26(7):507–534, 2001.
12. Shane Sendall. Combining Generative and Graph Transformation Techniques for Model Transformation: An Effective Alliance? In *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.
13. Kenneth Wacks. The successes and failures of standardization in home systems. In *2nd IEEE Conference on Standardization and Innovation in Information Technology*, 2001.
14. Roy Want, Trevor Pering, Gaetano Borriello, and Keith I. Farkas. Disapearing Hardware. *Pervasive Computing*, 1(1), 2002.
15. Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, Sept. 1991.
16. Ian Wilkie, Adrian King, Mike Clarke, Chas Weaver, and Chris Rastrick. *UML ASL Reference Guide*. Kennedy Carter Limited, 2001.



# Alignment and Maturity Are Siblings in Architecture Assessment

Bas van der Raadt, Johan F. Hoorn, and Hans van Vliet

Vrije Universiteit, Faculty of Exact Science,  
Information Management and Software Engineering,  
1081 HV Amsterdam, The Netherlands  
Fax +31 20 598 7718  
{bvdraadt, jfhoorn, hans}@few.vu.nl

**Abstract.** Current architecture assessment models focus on either architecture maturity or architecture alignment, considering the other as an explaining sub-variable. Based on an exploratory study, we conjecture that both alignment and maturity are equally important variables in properly assessing architecture organizations. Our hypothesis is that these variables conceptually differ, correlate, but do not explain one another. In this paper we describe our Multi-dimensional Assessment model for architecture Alignment and architecture Maturity (*MAAM*), which contains six main interrelated sub-variables that explain both alignment and maturity. We used existing models, literature from business and IS domains, and knowledge gained from previous research to identify the explaining variables. We constructed *MAAM* using structured modeling techniques. We are currently using a structured questionnaire method to construct an Internet survey with which we gather data to empirically validate our model. Our goal is to develop an architecture assessment process and supporting tool based on *MAAM*.

## 1 Introduction

Many large organizations suffer from the complexity of their business and IT structures and processes. This complexity makes it extremely difficult to keep an overview of the organization. Such overview, however, is needed to make important high-level strategic decisions about both business and IT priorities and activities. It also obstructs the identification of inconsistencies between business strategy and its supporting business and IT structures and processes, as well as possible bottle necks in those structures and processes. Furthermore, an organization needs a central point of reference to its business and IT situations in order to allow a meaningful communication about organizational decisions. Therefore these organizations are increasingly using architecture as a means of abstraction and communication, and as a management instrument to get a grip on their often incomprehensible business and IT situations, and the fit between the two [27].

The introduction and use of architecture within organizations, however, often does not go without a struggle. Applying architecture as a means to solve problems often also creates new problems. Therefore, the need is growing within organizations to assess the current architecture program and organization in order to identify the constraints and problems that hinder their success, and to specify the points of focus and a roadmap for improvement. Several assessment models have been constructed to individually evaluate either business-IT alignment (e.g., [6], [19]), or architecture maturity (e.g., [2], [10], [12], [25]). Both types of models indicate that there is a relationship between alignment and maturity. Maturity models see alignment as a sub-variable that (partly) explains the level of maturity of an architecture organization. Alignment models see maturity as such an explaining sub-variable. That there is a relationship between alignment and maturity also becomes evident from the fact that the two types of models both generally use the same factors to assess either maturity or alignment – variables such as governance, process, communication, scope, and partnership/involvement.

Unlike these existing assessment models, we see alignment and maturity as two conceptually different variables that do not explain one another. However, we do have strong indications that they correlate – from an exploratory study we observed that when architecture maturity increases, alignment generally improves too [27]. Also, in our view, only assessing either the alignment or maturity of an architecture organization is not enough. They should be used in combination, as two equally important variables, in order to properly assess an architecture organization.

Trying to improve an architecture organization is like driving a car. While driving a car, both velocity and direction are important variables. Without one of these, the destination of the journey cannot be reached. These two variables are conceptually different, but can nevertheless be related or affect one another. For example, when the car makes a turn, the driver is likely to slow down. This is because he or she wants to stay on the road or does not want to run into another car. When driving on an oval racing circuit, the same driver probably would not slow down. Although the two variables direction and velocity may be correlated or may have an impact on one another, changing direction does not explain why a car slows down. However, the driver's carefulness does. Viewed this way, velocity partially mimics architecture maturity and direction might symbolize architecture alignment. A poorly aligned architecture organization translates to a car that only turns right; it will end up driving in circles. By focusing on both business and IT aspects, and applying them properly, an architecture organization, quite like a car driver, is able to set out a course for maturity improvement and change direction when needed.

In this paper we introduce our Multi-dimensional Assessment model for architecture Alignment and architecture Maturity ( . . . ) that allows assessing an architecture organization on both architecture alignment and architecture maturity. . . . allows an assessor to better draw conclusions and identify points of improvement. We started constructing our model from the main variables

that explain both alignment and maturity in existing maturity and alignment assessment models. We extended these rather high-level variables by adding sub-variables using general theories from business and IS literature, which we adopted to the enterprise architecture domain, inspired by, among others, Henderson et al. and Chan. For example, Henderson et al. adopted the business theory of external positioning and internal arrangement for maximizing economical performance to the IT domain [16]. Chan found that the informal organizational structure is essential to IT alignment [5]. We introduced additional variables based on the architecture aspects and critical success factors we identified during an exploratory study at architecture active organizations, which is described in [27]. We used Structural Equation Modeling (SEM) techniques [29] to construct our model. We hope to validate our model using data gathered from the field through a structured questionnaire. The goal of this research is to develop an assessment process and tool support based on our findings.

This paper is structured as follows. In Section 2 we describe related work on business-IT alignment (Section 2.1) and architecture maturity (Section 2.2). In Section 3.1 we clarify how we measure both alignment and maturity of an architecture organization. We give a brief introduction to the six main variables in our model that explain both alignment and maturity in Section 3.2, and illustrate the depth of our model by describing in detail two of these variables, namely architecture governance, and organizational support for architecture activities. Section 3.3 shortly describes the practical application of the model. We end this paper with our conclusions in Section 4.

## 2 Related Work

### 2.1 Alignment

Since fundamental early work by Henderson and Venkatraman (e.g., [15], [16]), much is written about alignment in the literature. The notion and importance of alignment are well understood. Many definitions exist, but there is general consensus what alignment entails; the fit between business strategy, IT strategy, organizational structures and processes, and IT structures and processes (e.g., [5], [16], [18]). The goal of alignment is for IT activities to support those of the entire business [5].

Several alignment assessment models have been constructed. Luftman's strategic alignment assessment presents an approach for determining a firm's business-IT alignment based on six variables, namely communications, competency/value measurements, governance, partnership, skills, as well as scope and architecture [19]. This last variable is used to evaluate IT maturity, which indicates that Luftman sees the level of IT maturity as an explaining variable for the level of alignment. In [19], each of these six variables is assigned five levels of alignment. The model provides a short description of the aspects of each level. The level of alignment for each individual variable is determined by the answers to 6 or 7 questions. The model also describes the process of conducting an alignment

assessment. Luftman created this alignment assessment model based on his extensive research and practical experience. The model has been used to assess numerous Fortune 500 firms in order to fine tune and validate the model, and allows for cross-organizational benchmarking. Luftman's model is quite pragmatic, but ignores the interrelations between the variables that explain business-IT alignment, which our model does address. It also focuses on the general issues of business-IT alignment, rather than architecture specific issues, although there is much resemblance.

The Chief Information Officer (CIO) Council, a consortium of US Federal executive agency CIO's, developed an architecture specific alignment and assessment guide [6]. This guide describes a process which consists of three phases, the select phase, control phase, and evaluate phase. First, the select phase entails assessing business alignment; whether and to what degree a proposed investment aligns with business strategy. Second, in the control phase the technical alignment is assessed on how well the technology of investments aligns with the infrastructure architecture. Finally, the third phase evaluates both the architectural products and the architecture development process itself. This architecture assessment does not describe any core variables, which disables the identification of specific points of improvement, as well as interrelations between these variables. Also this assessment is quite specific to federal agencies, where our model will be applicable in assessing different types of organizations.

## 2.2 Maturity

Not so much fundamental research is done on architecture maturity. There is no real definition or clear description of architecture maturity in the literature. However, we could derive such a definition from other maturity studies in the field of IT, such as the SEI Capability Maturity Model [26]. Architecture maturity involves an organization's ability to organization-wide manage the development, implementation and maintenance of architectures on various levels – e.g. business, information systems, technical infrastructure, etc. Please note that with architecture maturity we focus on the entire architecture organization responsible for architecture development, and not merely on the architecture products they create, such as descriptions and models.

Also assessment models have been constructed to evaluate a firm's architecture maturity. These models come from two types of organizations, consulting firms such as Gartner [12] and METAGroup [2], and federal agencies, such as the US Office of Management and Budget (OMB) [25] and the US department of commerce (DoC) [10]. These models generally all work the same, in a way comparable to Luftman's alignment assessment model. They use a number of variables, ranging from 4 to 12, to assess architecture maturity. Typical variables are process, governance, communication, technology, alignment (business linkage), etc. The latter indicates that these models perceive the fit between business and IT to be an explaining variable for architecture maturity. Each variable knows five maturity levels, which are provided with a description of aspects. The individual level of maturity for each variable is based on answers to

generally 1 to 4 questions. Also, the assessment processes of these models share many similarities with that described by Luftman [19].

Assigning assessed organizations a level of architecture maturity or architecture alignment contains an element of danger. Reaching the next level of maturity or alignment could become a goal, although an assessment is meant as a means for improvement. This obsession of reaching the next level draws the attention away from the real important issues. Our model does not assign a maturity or alignment level to an assessed organization. It is able to give a more dynamic, multi-dimensional diagnosis that shows how the interrelations between variables influence and characterize the maturity and alignment of the architecture organization.

### 3 Assessing an Architecture Organization

In this section we explain how an architecture organization is assessed on its alignment and maturity using the model. We first define how these two variables are to be measured and in which way the sub-variables explain both architecture alignment and architecture maturity. In addition, we clarify why alignment and maturity covary.

#### 3.1 Alignment and Maturity

Architecture is a multi-dimensional phenomenon with different aspects originating from different fields of study, such as management and organization, business psychology, software architecture and engineering, knowledge management, and quality assurance. Maturity – the ability of an architecture organization to company-wide manage the development, implementation and maintenance of architectures on various levels – depends on how many of these aspects an architecture organization has identified as important and is using in performing its activities. Companies using only few aspects of architecture have a low maturity, and companies using many aspects have a high level of maturity [27]. Therefore, we assess architecture maturity by measuring how many aspects are being used by the architecture organization.

Alignment as the fit between business and IT is about business management and IT personnel communicating with each other and understanding each other. Thus, alignment comes from two sides. For example, a company is properly IT to business aligned when its IT personnel have business knowledge and are able to understand the business goals as well as create technological solutions to reach those goals. When a firm's senior (business) management knows what IT might offer them and is able to express their needs to IT personnel it is well business to IT aligned [5]. Therefore, in assessing alignment we measure how much IT knowledge business management and employees have, and how much IT management and personnel know about business issues.

From the above descriptions of alignment and maturity, it becomes clear that both are conceptually different, independent variables that characterize an ar-

chitecture organization. However, these two variables do correlate. For instance, some aspects of architecture specifically focus on business issues and others particularly on IT issues. When the number of architecture aspects that an architecture organization uses increases, an architecture organization becomes more mature. Another aspect of a mature architecture organization is the increasing exchange of architectural knowledge between different architecture functions, also between business-oriented and technical focused architecture functions, which makes them communicate and understand each other better. Therefore, with increasing maturity, business-IT alignment is likely to improve too.

Conversely, when alignment improves, the knowledge that senior (business) management have about IT increases, which makes them more aware of the opportunities IT offers them. The support of business executives boosts the priority of architecture-related projects. More money is invested in architecture, which results in more available architecture means – e.g., educating IT and business personnel, attracting experienced architects, acquiring or developing more architecture methods and techniques, etc. – and the presence of more architecture aspects. Therefore, alignment improvement results in an expected increasing maturity.

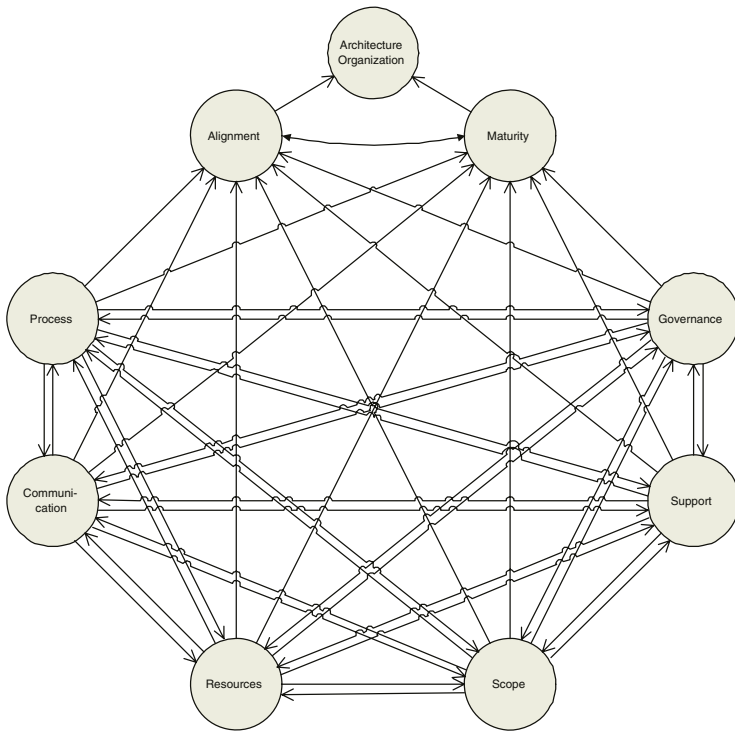
### 3.2

Based on existing models ([2], [19]), existing theories from business and IS literature (e.g., [5], [16]), and findings from exploratory research ([27]) we identified six key variables that explain both the level of maturity and the level of alignment:

- architecture development process
- architecture governance
- organizational support for architecture activities
- communication through and about architecture
- organizational and logical scope of architecture
- human and other architecture resources

Fig. 1 shows the top-level variables of [Fig. 1](#), how they are all interrelated, that they all explain both alignment and maturity, and that alignment and maturity correlate. In the figure, a single-headed arrow ( $\rightarrow$ ) from one variable to the other shows that the source variable explains the destination variable. A double-headed arrow ( $\leftrightarrow$ ) represents a correlation between two variables. These notations are taken from SEM, a framework for statistical analysis that also contains a variety of powerful analysis techniques we wish to use to validate our model.

In the remainder of this section we describe two variables in Fig. 1 into more detail, namely architecture governance and the organizational support for architecture activities. By describing these two variables in detail, we show the management and organization aspects and business psychological aspects of [Fig. 1](#), clarify how these variables explain each other, and we show the structure and depth of our model. The contributions of the low-level structures in Fig. 2 and Fig. 3 to the dependencies between these two top-level variables in Fig. 1 are

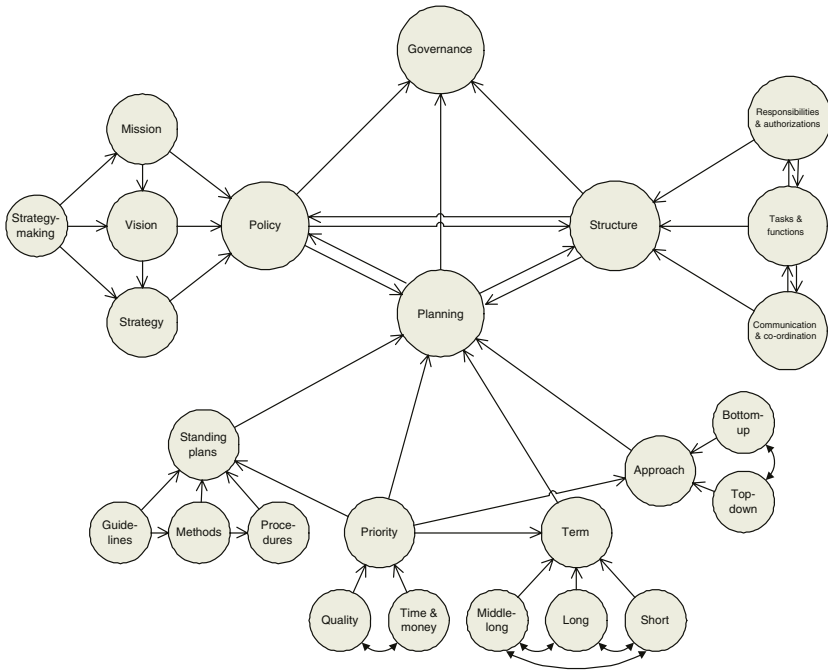


**Fig. 1.** The six main interrelated sub-variables that explain the two correlated variables alignment and maturity of an architecture organization

included in the discussion in the following notation: ( ... → ... ). An internal working paper [28] gives a full description of all six main variables of ... in a similar manner as in this section.

**Governance.** represents the management and organizational aspects of architecture. Like any organization, an architecture organization has to create a policy that states its mission, vision and strategy concerning architecture. In order to reach its strategic goals it has to structure and plan the activities of its architecture program. Fig. 2 shows the interrelated sub-variables that further explain the top-level variable ... in Fig. 1, using the same notation. The three main explaining, interrelated sub-variables of governance are: (1) architecture policy, (2) structure of the architecture organization, and (3) planning of architecture activities.

... An architecture organization needs to identify its environment (both internal and external to the company [16]), and clearly state its role, added value, and goals in a policy. An architecture policy consists of the mission, vision, and strategy concerning the architecture organization. The mission statement formu-



**Fig. 2.** The interrelated sub-variables that explain governance of the architecture organization

lates the justification of the architecture organization’s existence and its value to the business, its business partners, and the employees of the entire company. Further, the vision statement defines the goals and strengths of the architecture organization based on this mission statement. Finally, an architecture organization’s strategy lays the plans for accomplishing architectural goals; it puts the vision in action and clarifies goals and tactics. All three parts of the policy are a result of the strategy-making process [21]. The architecture strategy should be aligned with the business strategy of the entire company. Communicating the architecture mission [3], together with the vision and strategy of architecture [27] to those stakeholders improves organizational commitment and support ( . . . . . → . . . . .).

. . . . . When one employee cannot perform all activities of a company’s architecture organization, these activities need to be divided and structured. An organizational structure consists of three parts. Firstly, it shows the division of work into functions such as architects, and IT-managers, and their tasks. Secondly, it assigns authorizations and responsibilities to functions so that they are able to carry out their tasks. Thirdly, the organizational structure defines the communication and coordination means – e.g., work feedback, discussion and reports of progress, and coordination committees – to glue the divided work together [22].



The assignment of responsibilities and authorizations, and the introduction of communication and coordination means depend on the way work is divided into tasks and functions. After a while, existing communication and coordination means, as well as responsibilities and authorizations can become part of the culture of an organization; ‘the way work is done here’ ( *the way work is done here* → *the way work is done here* ).

The architecture organization’s structure is part of the structure of the entire organization; it is typically a staff or line department. It should also be a reflection of the architecture policy and be aligned with the policy and structure of the entire firm. Ideally structure follows strategy [4], but in practice making a new policy requires paying attention to existing organization structures and operational processes [22]. In this the architecture organization is no exception.

..... An architecture organization needs to plan its activities. Planning is the process of information processing that results in decisions about and coordination of future acts so that these acts can be controlled. Decisions about future activities can be made on long, middle-long, and short term. An important short term goal is communicating the added value of architecture to senior and middle management [19]. Improving the quality and structure of information systems and infrastructure is typically a long-term goal [27]. The architecture organization’s planning should serve the architecture strategy [23], but strategy also depends on the available planning means. When short-term goals are emphasized, middle-long and long term goals will be influenced negatively, hence the covariance between the three different variables.

Clearly assigning priority to either the quality of architecture products or the availability of resources, such as time and money, may prevent many problems. Architects prioritize quality because they are responsible for the quality of a design. Management, however, is more likely to prioritize the use of resources because they are responsible for finishing projects within time and budget. In practice this difference in responsibilities and prioritizing often results in tension between the two groups [27]. The choice of prioritizing quality has a negative correlation with the use of time and money, and vice versa.

The planning process knows two approaches. The first is a top-down approach that starts with senior management initiating the planning of architecture activities. This approach greatly depends on senior management’s ability to get operational support. The second is a bottom-up approach where the planning of architecture activities starts within the departments or divisions; they seek senior management support ( *bottom-up approach* → *top-down approach* ). In practice, the planning and execution of architecture activities often combines both approaches [27].

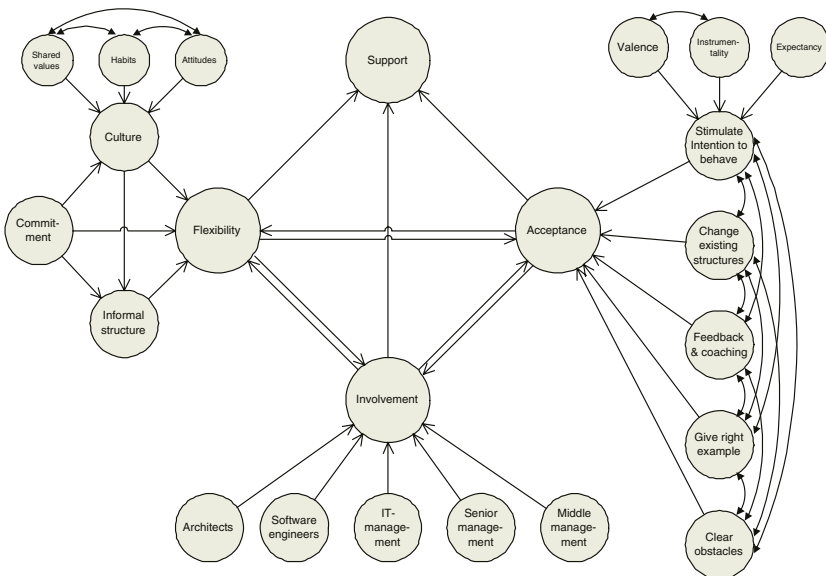
Standardization of the architecture processes is accomplished by making standing plans. Three kinds of standing plans exist: policy guidelines, standard methods, and standard procedures. Firstly, policy guidelines are general indicators of expected behavior or decisions in general situations. Secondly, standard methods are a refinement and specification of policy guidelines. They in-

dicating how to deal with specific situations. Thirdly, standard procedures prescribe the range of connected tasks that form a unity with a precise rounded off outcome.

The planning of the architecture activities should reflect the architecture policy. Planning – a means of coordination of future activities – depends on the structure it has to coordinate, but the organizational structure also depends on the availability of means for planning those structured activities.

**Support.** characterizes the psychological and social aspects of architecture. Introducing and using architecture inflicts organizational changes, which can only be successfully carried through with organizational support. Fig. 3 depicts the interrelated sub-variables that more specifically explain the key variable ‘‘Support’’ in Fig. 1. Organizational support depends on three variables: (1) organizational acceptance of architecture-driven changes, (2) the flexibility of an organization in adjusting to its environment, and (3) organizational involvement in the architecture program.

Changing organizational behavior requires changing the organizational culture, which refers to the shared values and opinions, habits, and attitudes in an organization [9] and is typically difficult to change [17], [24]. A culture’s compatibility with the intended changes influences the ability to make those changes.



**Fig. 3.** The interrelated sub-variables that explain organizational support for architecture activities

In theory, changing the structure of an organization is quite simple, but in practice it often has unexpected repercussions because of the existing informal structure (e.g.,  $Structure \rightarrow Informal\ Structure$ ). The informal structure is the behavior of organization members, which is not explicitly structured in advance. It consists of informal networks of personal contacts. Most companies see informal networks as a burden rather than an aid. Senior executives often work around them, or try to ignore them [7]. This hinders acceptance of decisions and slows change down. When it comes to realizing alignment, informal structures play a vital role [5].

Organizational commitment – the degree to which employees feel connected to the organization – is an important explaining variable of organizational flexibility. Personnel might completely internalize organizational values [24], which makes culture difficult to change. They may also conform to only typical behavior without the underlying values. This group works according to the rules, which are easy to change, and does not care about the shared values, which are hard to change. Finally, organization members could share the central values of an organization and at the same time criticize those values when needed. This group is open for innovation and (architecture-driven) changes. It is important to involve these innovative employees in architecture activities.

Commitment could be a shared value and be part of an organization culture. If commitment is low, an organization member is more likely to go against the rules and use the informal structure to get work done. Working by the rules might also be a shared value – especially in hierarchical organizations – which disables the arise and use of an informal structure (e.g.,  $Commitment \rightarrow Informal\ Structure$ ).

Change. In order to carry through architecture-driven changes, old organizational behavior needs to be transformed to new behavior. Many phase models for changing organizational behavior exist (e.g., [9], [17]). They show many similarities. First of all, the change-initiators – e.g., architects, IT-managers, or senior management – should give the right example of how things should be done in the future. Secondly, they should motivate organization members in adopting new behavior by creating expectations and making clear what value these changes can be to them. Expectancy-theory may be used to measure motivation and contains three variables: (1) the expectancy that certain behavior leads to a certain result, (2) the instrumentality that a certain result leads to a certain reward, and (3) the valence (subjective appreciation) of that reward [30]. Thirdly, change-initiators should give feedback about wrong, old behavior, and coach employees in new behavior. The fourth variable is clearing obstacles that thwart change, like a manager on a key position who is unwilling to change, or the old reward system that stimulates old behavior. Therefore it is important to change these existing structures so that they stimulate new behavior and prevent old behavior (e.g.,  $Structure \rightarrow Informal\ Structure$ ). All these above factors should be combined into one change program for the architecture program. Therefore they are likely to covary.

Acceptance greatly depends on whether the intended changes will alter the balance of power in the organizational structure. If so, people who will lose power because of the architecture-driven changes are unlikely to cooperate, and vice versa ( . . . . . → . . . . . ).

. . . . . The involvement of key organization members – senior management [18], line management, IT-management, software engineers, and architects [2] – is critical to the architecture program’s success. For example, senior management support for an architecture program would automatically create support by a large part of the organization, because of their leadership position.

Architecture might have its origin in different parts of an organization, either at the IT department, or at business management. If architecture has its origin at senior management, middle management is vital in passing on the architecture program to the operational level where the architecture should be implemented. If the architecture program is an initiative of the IT-department, IT-management and architects play an important role in selling architecture to senior management and the rest of the organization [27].

The involvement of software engineers is vital because they build the software systems architects design. The methods and techniques architects use and the quality of the architectures they produce play an important role in getting the software engineers to accept that architecture. If architects use practically oriented methods and techniques that result in comprehensible designs that fit the practice of implementing software systems, software engineers will be more likely to accept those designs [27].

Because all groups have such different interests, it is unlikely that their levels of involvement covary. For instance, a high level of involvement of software engineers – who are developing software according to the architecture – does not result in a higher senior management involvement. Every group has another reason to adopt or reject the architecture program, so they all need separate attention in raising their involvement.

### 3.3 Assessment Process

We are planning to gather data through a questionnaire (for an example, see Appendix), which allows us to validate . . . . . In our assessment process, the same questionnaire is used to gather data among employees within one department, a division, an entire firm, or even across enterprises, to assess the involved architecture organization(s). In order to visualize the results of each individual questionnaire in a supporting tool we use the Architecture Alignment Model ( . . . ), introduced in [27]. This model relates architecture maturity on the horizontal axis with architecture alignment on the vertical axis. If clustering the results of the individual questionnaires shows groups with large differences when it comes to alignment, efforts to re-align these groups should receive high priority. Combining all results into an overall architecture assessment allows benchmarking an architecture organization with competitors, which might indicate the necessity to improve architecture maturity in order to keep up with them.

## 4 Conclusion

In this paper we indicate the importance of a multi-dimensional approach to assess an architecture organization. We introduce our Multi-dimensional Assessment model for architecture Alignment and architecture Maturity ( . . . ) that is able to establish the current situation of a firm's architecture organization, identify the points of improvement, and construct a plan to address these points.

Existing architecture maturity assessment models assume that architecture alignment explains architecture maturity. On the other hand, architecture alignment assessments see architecture maturity as an explaining variable for architecture alignment. This indicates that there is an interrelation between the two. We view architecture alignment and architecture maturity as conceptually different. Also, in our view architecture alignment is not an explaining factor of architecture maturity, and vice versa. Our hypothesis is that they correlate. When architecture maturity increases, architecture alignment generally increases too, and vice versa.

To construct our model, we used variables of existing assessment models ([2], [19]), theories from literature from various research fields adopted to the IT domain (e.g., [5], [16]), and previous research on the aspects and critical success factors of architecture in practice [27]. Our model contains six main interrelated variables that all explain each other and both alignment and maturity. These six variables are again individually explained by other sub-variables.

We are currently constructing a self-administered Internet survey, using a structured questionnaire method [11], to internationally gather data at architecture active organizations of all types and sizes. Using these data, we hope to validate . . . using SEM analysis techniques. This involves validating the hypothesis that alignment and maturity correlate and do not explain each other, and determining the strength of the relationships between the sub-variables that explain both alignment and maturity as well as how these two explain the architecture organization. An empirical study within the business-IT alignment domain with an approach related to ours is conducted by Croteau et al. [8].

## References

1. Bass, L., Clements, P., and Kazman, R.: *Software Architecture in Practice (2nd ed.)*. Addison-Wesley (2003)
2. Burke, B.: Evolving Architecture Maturity, Enterprise Planning and Architecture Strategies. *EPAS META Practices*, No. 67. META Group, Inc. (2002)
3. Campbell, A., and Tawadey, K.: *Mission and Business Philosophy; Winning Employee Commitment*. Heinemann Professional Publishing (1990)
4. Chandler, A.D.: *Strategy and structure: Chapters in the history of the industrial enterprise*. MIT Press (1962)
5. Chan, Y.: Why haven't we mastered alignment? The importance of the Informal Organization Structure. *MIS Quarterly Executive* Vol. 1, No. 21. (2002) 76–112
6. Federal Architecture Working Group: *Architecture Alignment and Assessment Guide*. The Federal Chief Information Officers Council (2000)

7. Cross, R., and Prusak, L.: The People Who Make Organizations Go—or Stop. *Harvard Business Review*, June. (2002) 104–112
8. Croteau, A.M., and Bergeron, F.: An information technology trilogy: business strategy, technological deployment and organizational performance. *Journal of Strategic Information System*, Vol. 10. Elsevier (2001) 77–99
9. Dawson, S.: *Analysing organisations (3rd ed.)*. Macmillan, London (1996)
10. The Department of Commerce Enterprise IT Architecture Advisory Group: *IT Architecture Capability Maturity Model*. Department of Commerce, USA (2003)
11. Dillman, D.A.: *Mail and Internet Surveys: The Tailored Design Method (2nd ed.)*. John Wiley and Sons (1999)
12. Gartner Consulting: *Architecture Maturity Assessment Evaluation*. Gartner Inc. (2002)
13. Guttman, L.: A faceted definition of intelligence. *Studies in Psychology*, Vol. 14. (1965) 166–181
14. Harter, D.E., and Slaughter, S.A.: Process Maturity and Software Quality: a Field Study. *Proceedings of the 21th International Conference on Information Systems (ICIS2000)*. Association for Information Systems (2000) 407–711
15. Henderson, J.C., and Venkatraman, N.: *Strategic Alignment: A Framework for Strategic Information Technology Management*. Center for Information Systems Research Working Paper No. 190. Massachusetts Institute of Technology, Cambridge, MA (1989)
16. Henderson, J.C., and Venkatraman, N.: Strategic Alignment: Leveraging Technology for Transforming Organizations. *IBM Systems Journal*, Vol. 32, No. 1. (1993) 4–16
17. Kotter, J., and Heskett, J.: *Corporate Culture and Performance*. The Free Press, New York (1992)
18. Luftman, J.N., Lewis, P.R., and Oldach, S.H.: Transforming the Enterprise: The Alignment of Business and Information Technology Strategies. *IBM Systems Journal*, Vol. 32, No. 1. (1993) 198–221
19. Luftman, J.N.: Assessing Business-IT Alignment Maturity. *Communications of AIS*, Vol. 4, Article No. 14. Association for Information Systems (2000)
20. META Group, Inc.: Architecture Capability Assessment. *META Practice*, Vol. 4, No. 7. META Group, Inc. (2000)
21. Mintzberg, H.: Crafting Strategy. *Harvard Business Review*, July-Aug. (1987) 66–75
22. Mintzberg, H.: *The Structuring of Organizations*. Prentice Hall (1979)
23. Mintzberg, H.: The Fall and Rise of Strategic Planning. *Harvard Business Review*, Jan-Feb. (1994) 107–114
24. Neuijen, J.A.: *Diagnosing Organizational Cultures*. Wolters-Noordhoff (1992)
25. OMB FEA Program Management Office: *OMB Enterprise Architecture Assessment v1.0 Guidelines*. The Office of Management and Budget, The Executive Office of the President, USA (2004)
26. Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, C.V.: Capability Maturity Model, Version 1.1. *IEEE Software*, Vol. 10, No. 4. IEEE Computer Society Press (1993) 18–27
27. van der Raadt, B., Soetendal, J., Perdeck, M., and van Vliet, H.: Polyphony in Architecture. *Proceedings 26th International Conference on Software Engineering (ICSE2004)*. IEEE Computer Society (2004) 533–542
28. van der Raadt, B., Hoorn, J.F., and van Vliet, H.: *Assessing Architecture Alignment and Architecture Maturity*. IMSE Internal Working Paper. Vrije Universiteit, Amsterdam, the Netherlands (2004)

29. Rigdon, E.E.: Structural equation modeling. *Modern methods for business research* G. Marcoulides (ed.). Lawrence Erlbaum, Mahwah, NJ (1998) 251–94
30. Vroom, V.H.: *Work and motivation*. Wiley, New York. (1964)

## Appendix: Examples of Alignment and Maturity Scales

Below we give two example scales to measure alignment and maturity using Likert items with a rating scale from 0 to 5 ('completely disagree' to 'completely agree'). We plan to test these scales on their psychometric quality by checking the convergent and discriminant validity of their items. Ultimately, we plan to use a combination of analysis techniques (correlation, regression, factor analysis, path analysis, and model fit [11]) to validate .

---

### Architecture Alignment Scale:

- item  $A_1$ : Business and IT strategy have the same priorities.  
 item  $A_2$ : Senior management is involved in developing both strategies.  
 item  $A_3$ : Business and IT strategy facilitate one another.  
 item  $A_4$ : Business and IT managers understand each other.

- item  $A_5$ : Business and IT strategy have different priorities.  
 item  $A_6$ : Senior management is involved in establishing only one strategy.  
 item  $A_7$ : Business and IT strategy constrain one another.  
 item  $A_8$ : Business and IT managers misunderstand each other.
- 

### Architecture Maturity Scale:

- item  $M_1$ : Business architecture is well established.  
 item  $M_2$ : IT architecture is well developed.  
 item  $M_3$ : Business architecture is accepted.  
 item  $M_4$ : People approve of the IT architecture.

- item  $M_5$ : Business architecture is badly established.  
 item  $M_6$ : IT architecture is ill-developed.  
 item  $M_7$ : People reject the business architecture.  
 item  $M_8$ : People oppose to the IT architecture.
-

# Verification of EPCs: Using Reduction Rules and Petri Nets

B.F. van Dongen, W.M.P. van der Aalst, and H.M.W. Verbeek

Department of Technology Management, Eindhoven University of Technology,  
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands  
{b.f.v.dongen, w.m.p.v.d.aalst, h.m.w.verbeek}@tm.tue.nl

**Abstract.** Designing business models is a complicated and error prone task. On the one hand, business models need to be intuitive and easy to understand. On the other hand, ambiguities may lead to different interpretations and false consensus. Moreover, to configure process-aware information systems (e.g., a workflow system), the business model needs to be transformed into an executable model. Event-driven Process Chains (EPCs), but also other informal languages, are intended as a language to support the transition from a business model to an executable model. Many researchers have assigned formal semantics to EPCs and are using these semantics for execution and verification. In this paper, we use a different tactic. We propose a two-step approach where first the informal model is reduced and then verified in an interactive manner. This approach acknowledges that some constructs are correct or incorrect no matter what interpretation is used and that the remaining constructs require human judgment to assess correctness. This paper presents a software tool that supports this two-step approach and thus allows for the verification of real-life EPCs as illustrated by two case studies.

## 1 Introduction

Nowadays, information systems such as Workflow Management (WFM) [4, 15] and Enterprise Resource Planning (ERP) [12] systems are used to support a wide range of operational business processes. These systems are often configured on the basis of a process model and therefore it is of the utmost importance that the process model is correct. Therefore, many researchers have worked on the verification of process definitions. Several tools and approaches have been developed for the workflow domain. The basis for most of the work in this area is typically the construction of mathematically sound and executable semantics for a specific modeling language. However, when looking at process modeling techniques, we see that very often, such semantics do not exist, or are too complex for a process designer to comprehend. Still, creating models in these languages is usually easy to do and the resulting models are understood by a broad audience. In this paper, we will focus on one of these modeling languages: Event-driven Process Chains (EPCs) [11, 12, 22]. EPCs are used in a large variety of systems, most notably SAP/R3, the Aris Toolset and Aris



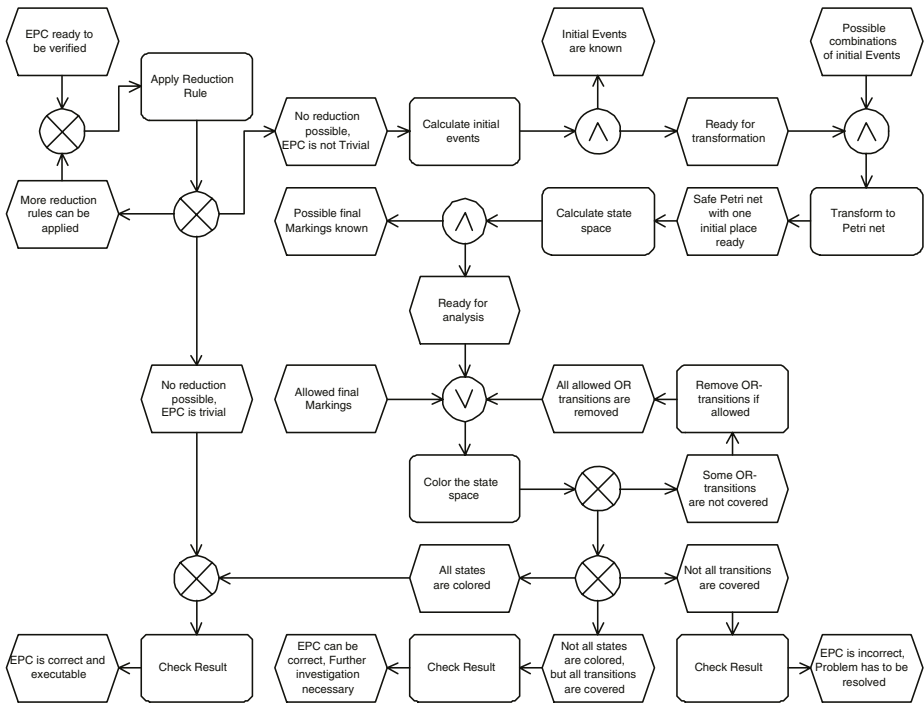


Fig. 1. EPC describing the EPC verification process

Process Performance Monitor (PPM). We will . . . provide “yet another formal semantics” for EPCs to be used as a basis for verification. Instead, we look at verification from a designer point of view. We help the designer to find structural conflicts, and give feedback about possible semantical problems. Furthermore, if there is a trivial executable semantics, we will also provide this information to the designer. However, in case of possible semantical problems, we leave the designer in charge and let him decide what to do.

Since this paper is about the verification of EPCs, we use an EPC to describe our approach. Figure 1 shows the details of the verification process in terms of an EPC. The process consists of two main parts. First we take the EPC that is defined by a process designer and, using simple reduction rules, we extract the possibly problematic area. Then we translate the result into a Petri net and use variants of existing Petri-net-based verification techniques to give feedback to the designer.

As we discussed before, we will look at verification from a designers perspective, instead of from a formal perspective. Therefore, we will look at a more relaxed correctness notion (similar to relaxed soundness [8, 7]). This process consists of multiple steps. First, we introduce fictive nodes to make an EPC with . . . initial event and . . . final event. Then, we use some reduction rules to eliminate the “easy” constructs for which we know that they are correct.

In Section 4 we will discuss these reduction rules. If the EPC under investigation reduces to the trivial EPC, it is correct. In the reduction step, all functions and events except initial and final event are removed. Furthermore, local choices and trivial synchronization constructs are eliminated. If the result of the reduction is not the trivial EPC, the next step is to translate the EPC into a Petri net in a rather simplistic way. In the last step we use the theory of workflow nets [2, 4] and its related concepts such as soundness [2] and relaxed soundness [8, 7]. The last step will provide the designer with one of the following three answers:

**The Petri net is *sound***, which means that the original EPC is correct and no further reviewing is necessary.

**The Petri net is *relaxed sound***, which means that some constructs need further reviewing of the designer.

**The Petri net is *not relaxed sound***, which means that there are unrecoverable problems with the EPC. Corrections are necessary to create a correct EPC.

We have developed a tool for the analysis of EPCs using the approach depicted in Figure 1. The tool is implemented in the context of the ProM framework<sup>1</sup>.

In the remainder of this paper we will first look at some related work in Section 2. Then, in Section 3 we introduce concepts like EPCs, Petri nets, soundness, and relaxed soundness. As mentioned before, Section 4 introduces a set of powerful but simple reduction rules for EPCs. In Section 5 we translate the EPC into a Petri net and discuss the verification process in more detail. In Section 6, we briefly describe two case studies: one involving the trade process within a large Dutch bank and the other involving the SAP R/3 reference models. Section 7 concludes the paper.

## 2 Related Work

Since the mid-nineties, a lot of work has been done on the verification of process models, and in particular workflow models. In 1996, Sadiq and Orłowska [19] were among the first ones to point out that modeling a business process (or workflow) can lead to problems like livelock and deadlock. In their paper, they present a way to overcome syntactical errors, but they ignore the semantical errors. Nowadays, most work that is conducted is focusing on semantical issues, i.e. “will the process specified always terminate” and similar questions. The work that has been conducted on verification in the last decade can roughly be put into three main categories. In this section, we present these categories and give relevant literature for each of them.

---

<sup>1</sup> See [www.processmining.org](http://www.processmining.org) for details.

## 2.1 Verification of Models with Formal Semantics

In the first category we consider the work that has been done on the verification of modeling languages with formal semantics. One of the most prominent examples of such a language are Petri nets [9, 17, 18]. Since Petri nets have a formal mathematical definition, they lend themselves to great extent for formal verification methods. Especially in the field of workflow management, Petri nets have proven to be a solid theoretical foundation for the specification of processes. This, however, led to the need of verification techniques, tailored towards Petri nets that represent workflows. In the work of Van der Aalst and many others [2, 6, 8, 10, 23] these techniques are used extensively for verification of different classes of workflow definitions. However, the result is the same for all approaches. However, not all modeling languages have a formal semantics. On the contrary, the most widely used modeling techniques, such as UML and EPCs are merely an informal representation of a process. These modeling techniques therefore require a different approach to verification.

## 2.2 Verification of Informal Models

Modeling processes in a real-life situation is often done in a less formal language. People tend to understand informal models easily, and even if models are not executable, they can help a great deal when discussing process definitions. However, at some point in time, these models usually have to be translated into a specification that can be executed by an information system. This translation is usually done by computer scientists, which explains the fact that researchers in that area have been trying to formalize informal models for many years now. Especially in the field of workflow management, a lot of work has been done on translating informal models to Petri nets. Many people have worked on the translation of EPCs to Petri nets, cf., [1, 3, 7, 14]. The basic idea of these authors however is the same: "Restrict the class of EPCs to a subclass for which we can generate a sound Petri net". As a result, the ideas are appealing from a scientific point of view, but not useful from a practical point of view.

Also non-Petri-net based approaches have been proposed for the verification of informal modeling languages. One of these ideas is graph reduction. Since most modeling languages are graph-based, it seems a good idea to reduce the complexity of the verification problem by looking at a reduced problem, in such a way that correctness is not violated by the reduction, i.e. if a model is not correct before the reduction, it will not be correct after the reduction and if the model is correct before the reduction, it will be correct after the reduction. From the discussion on graph reduction techniques started by Sadiq and Orłowska in 1999 [20, 21] and followed up by many authors including Van der Aalst et al. in [5] and Lin et al in [16], it becomes clear that again the modeling language is restricted to fit the verification process. In general this means that the more advanced routing constructs cannot be verified, while these constructs are what makes informal models easy to use.

The tendency to capture informal elements by using smarter semantics is reflected by recent papers, cf. [3, 7, 13]. In these papers, the problem is looked at from a different perspective. Instead of defining subclasses of models to fit verification algorithms, the authors try to give a formal semantics to an informal modeling language. Even though all these authors have different approaches, the goal in every case is similar: *to capture the informal elements of a modeling language in a formal semantics that can be used for verification*.

### 2.3 Verification by Design

The last category of verification methods is somewhat of a by-stander. Instead of doing verification of a model given in a specific language, it is also possible to give a language in such a way that the result is always correct. An example of such a modeling language is IBM MQSeries Workflow [15]. This language uses a specific structure for modeling, which will always lead to a correct and executable specification. However, modeling processes using this language requires advanced technical skills and the resulting model is usually far from intuitive.

In this section, we have presented an overview of the literature on process model verification. We have categorized the various methods in three main categories and pointed out why many of them are not used in practice. The main difference between the technique presented in this paper and existing literature is that we will not restrict an informal modeling language to fit our verification, nor will we give an executable specification of an informal model. Instead, we combine the best of existing literature and provide a system designer with a tool to find possible problems in a specification. We do not aim at solving these problems. Instead, we assume the designer to be able to decide whether or not a specification is correct. The result of our work will be a verification plug-in, implemented in the Process Mining (ProM) Framework, that is able to import EPCs defined in the Aris Toolset<sup>2</sup> and will provide the designer with feedback about possible problems.

## 3 Preliminaries

In this section, we introduce some basic concepts needed for the verification process. We introduce the modeling language of EPCs and Petri nets. Furthermore, we introduce the notion of soundness and relaxed soundness of Petri nets.

### 3.1 Event-Driven Process Chains

The concept of Event-driven Process Chains is to provide an intuitive modeling language to model business processes. They were introduced by Keller, Nüttgens and Scheer in 1992 [11]. It is important to realize that the language is not intended to be a formal specification of a business process.

---

<sup>2</sup> See [www.ids-scheer.com](http://www.ids-scheer.com) for information about the ARIS toolset.

An EPC consists of three main elements. Combined, these elements define the flow of a business process as a chain of events. The elements used are:

**Functions**, which are the basic building blocks. A function corresponds to an activity (task, process step) which needs to be executed. A function is drawn as a box with rounded corners.

**Events**, which describe the situation before and/or after a function is executed. Functions are linked by events. An event may correspond to the position of one function and act as a precondition of another function. Events are drawn as hexagons.

**Connectors**, which can be used to connect functions and events. This way, the flow of control is specified. There are three types of connectors:  $\wedge$  (and),  $\times$  (xor) and  $\vee$  (or). Connectors are drawn as circles, showing the type in the center of the circle.

Functions, events and connectors can be connected with edges in such a way that (i) events have at most one incoming edge and at most one outgoing edge, but at least one incident edge (i.e. an incoming or an outgoing edge), (ii) functions have precisely one incoming edge and precisely one outgoing edge, (iii) connectors have either one incoming edge and multiple outgoing edges, or multiple incoming edges and one outgoing edge, and (iv) in every path, functions and events alternate (no two functions are connected and no two events are connected, not even when there are connectors in between.)

From the definition of an EPC it is clear that a process always starts when a certain event occurs. Such an event should be one of the events without incoming edges. After the process is finished, the events that have not been dealt with yet should be events without outgoing edges. If this is the case, we call the EPC

### 3.2 Petri Nets

Petri nets are a formal language that can be used to specify processes. Since the language has a formal and executable semantics, processes modeled in terms of a Petri net can be executed by an information system. In this paper, we use a variant of the classic Petri-net model, namely Place/Transition nets. For an elaborate introduction to Petri nets, the reader is referred to [9, 17, 18]. A Petri net consists of two modeling elements:

**Transitions**, which typically correspond to either an activity (task, process step) which needs to be executed, or to a “silent” step that takes care of routing.

**Places**, which are used to define the preconditions and postconditions of transitions. A transition can be *fired* (executed) if the precondition is satisfied. The result of such a firing will be that the postcondition holds.

Transitions and places are connected through directed arcs in such a way that (i) places and transitions have at least one incident edge and (ii) in every path, transitions and places alternate (no place is connected to a place and no transition is connected to a transition.)

To denote the state a process execution the concept of tokens is used. A token is placed inside a place to show that a certain condition holds. Each place can contain arbitrarily many of such tokens. If a transition execution occurs (or  $\dots$ ), one token is removed from each of the input places and one token is produced in each of the output places. This restricts the behavior in such a way that a transition can only occur when there is at least one token in  $\dots$  of the input places. The distribution of tokens over the places is called a  $\dots$ , or a  $\dots$ .

In this paper, we mostly consider Workflow nets (WF-nets). WF-nets are a subclass of Petri nets tailored towards workflow modeling and analysis. A WF-net has one source place and one sink place and all transitions are on a path from source to sink. Based on WF-nets correctness notions such as soundness [2, 4], generalized soundness [10] and relaxed soundness [8, 7] have been defined.

### 3.3 State Space

Petri nets can be used as executable specifications of business processes. Whenever a Petri net is given, together with an initial marking it is possible to capture all possible behavior in a  $\dots$ . The only caveat here is that the Petri net should be constructed in such a way that there is a maximum number of tokens that can appear in a place. This property is called  $\dots$ , and a special case is when the maximum number of tokens in each place is one. In that case this is called  $\dots$ .

In this section, we introduced EPCs and Petri nets. In the remainder of this paper, we show the process of EPC verification. The first step is made in Section 4, where we reduce the verification problem of a large EPC to that of a smaller EPC. In Section 5, we use Petri nets and state spaces to decide whether the EPC is correct.

## 4 Reduction Rules

In general, EPCs can contain a large number of functions, events and connectors. However, for the verification of EPCs, not all of these elements are of interest. In particular, we are interested in the routing constructs that are used in the EPC, since that is where the errors can be. Furthermore, it is obvious that some constructs are trivially correct, for example if a split of some type is followed by a join of the same type. In this section, we introduce a set of reduction rules. These rules can be applied on any EPC in such a way that, if the EPC is correct before the reduction, then the result after reduction is correct and if the EPC is not correct before reduction, then the result after reduction is not correct, i.e. these rules are  $\dots$ . However, we do not intend these rules to be complete. Instead, they merely help to speed up the verification process, by removing trivial parts before going to the more complex steps in terms of computation time.

It is easily seen that the applying the reduction rules does not result in an EPC, since functions and events no longer alternate. However, for the process of

verification, this is not a problem and we will refer to this reduced model as a

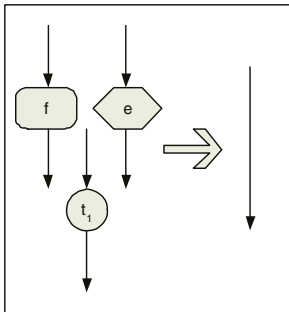


Fig. 2. Trivial construct

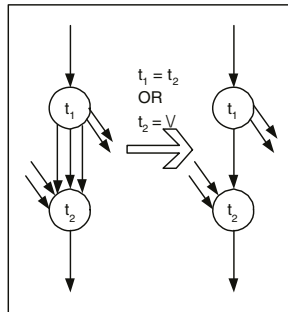


Fig. 3. Simple split/join

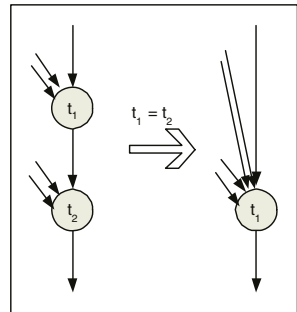


Fig. 4. Similar joins

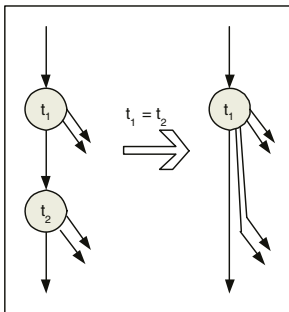


Fig. 5. Similar splits

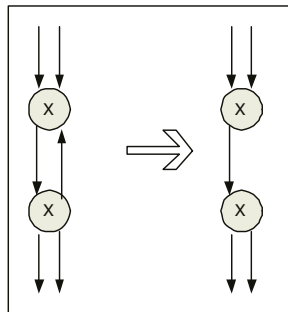


Fig. 6. XOR loop

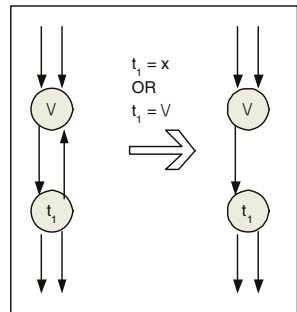


Fig. 7. optional OR loop

Figure 2 shows the reduction rule for trivial constructs. It shows that a function  $f$ , an event  $e$  or a connector with type  $t_1$  with precisely one ingoing and one outgoing edge can be removed completely. As stated before, we are only interested in routing constructs and functions, events or connectors with only one incoming and only one outgoing edge do not provide any routing information. Therefore, they can be removed while preserving correctness.

Figure 3 shows the reduction rule for a split that is followed by a join connector. This rule can be applied if both connectors are of the same type (i.e. AND, OR or XOR), or if the join connector is of type OR. Again it is trivial to see that correctness is preserved.

Figures 4 and 5 show the rules for two connectors of the same type that directly follow each other. These two connectors can then be merged into one connector of the same type. Note that syntactical restrictions of (reduced) EPCs do not allow for more than one edge between the first and the second connector, since connectors are either a split or a join and never both.

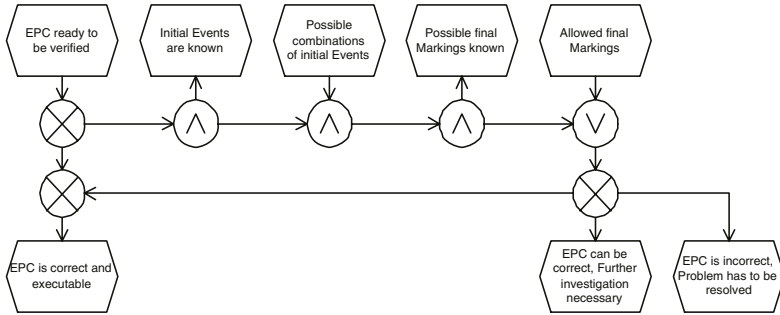


Fig. 8. Reduced EPC for the verification process

Finally, figure 6 and 7 show two very similar reduction rules that deal with loops. In these cases correctness preservation is less straightforward. For Figure 6 it is clear that removing the possibility to loop back is correctness preserving, because the “backward arc” does not introduce any new states. Figure 7 shows an optional rule. Unlike the others it is not correctness/incorrectness preserving in any situation like the first five rules. The rule assumes that the intended semantics is safe (i.e., no multiple activations of functions and no events that are marked multiple time). This implies that if  $t_1$  is an OR-join either the backward arc is taken or any combination of the other arcs.

Figure 8 shows the result of applying reduction rules to the EPC of Figure 1. The resulting reduced EPC does not contain any functions, and only some of the connectors from the original EPC. We know that none of the reduction rules will make the reduced EPC incorrect if the original was correct, and they will not make the reduced EPC correct if the original was incorrect. Therefore, we can now proceed with the verification process using this reduced EPC and the result can directly be translated back to the original EPC.

## 5 Verification of the Reduced EPC

In the previous section, we introduced reduction rules for EPCs in such a way that we can use a reduced EPC for the verification process. In this section, we will translate the reduced EPC into a safe Petri net (i.e. a Petri net where a place contains at most one token). This is also the part of the verification process where user interaction plays an important role. The user has to provide us with possible combinations of initial events. These combinations are then translated into initial markings of the Petri net. By calculating the state space, we can then provide the user with all possible combinations of final events that can happen. It is again up to the user to divide those into a set of desired and undesired combinations. Using this information we go into the final stage, where we use a simple coloring algorithm on the state space to decide whether the reduced EPC is correct. This is then translated back to the original EPC.



The whole process of verification described in this section is implemented in our the ProM framework. This tool interacts with the Aris toolset, which is widely used in industry for modeling business processes.

**User Interaction 1.** As we stated before, the process of EPC verification relies on user interaction at two points. The first point is where the user has to specify which combinations of initial events can appear to initiate the process described by the EPC. Using this information from the user, we can calculate which initial markings are possible for the Petri net that we will build. If we consider the example from Figure 1, then there is only one combination of events that can start the process. This is the combination of the events “EPC ready to be verified”, “Possible combinations of initial events” and “allowed final markings”. It has to be noted that the events “Possible combinations of initial events” and “allowed final markings” can only appear as a consequence of some choice that was made in the model. However, these causalities are not expressed in the EPC, and therefore they cannot be known to the verification system. As can be seen in the procedure shown in Figure 1, we are now ready to transform the EPC into a Petri net.

**Translation to Petri Net.** Many authors have described algorithms to translate EPCs to Petri nets. In this paper, we use a modified version of the translation proposed in [8, 7]. The translation presented there gives a translation into normal Petri nets, whereas we use the same translation algorithm, but assume the result to be a safe Petri net, or elementary net. In terms of an EPC, this corresponds to ruling out the situation where an event can occur more than once before it is dealt with. Converting an elementary net into a Petri net again is a trivial step, since it only requires the duplication of all places. The choice for elementary nets is motivated by the idea that an EPC should clearly reflect its behavior from its design. When one event is allowed to appear again, before it is dealt with by some function, this does not hold any more. The result of the transformation process is shown in Figure 9. Note that in the layout of the Petri net the reduced EPC from Figure 8 is visible.

Using the combinations of initial events calculated in the previous step, we are ready for the state space generation.

**State Space Generation.** As we stated in Section 3.3, it is possible to calculate the entire state space for a Petri net, if it is bounded, and the Petri net is not too large. In our case, the Petri net is likely to be of limited size, since we used the reduction rules to get a model that is as small as possible. Furthermore, the Petri net contains at most one token in each place. Therefore we are likely to be able to calculate the state space.

**User Interaction 2.** Now that we have calculated the state space, we are able to provide the user with details about the possible outcomes of the process. In our example, there are many different outcomes that were not intended to be there. The reason for this is in the informal definition of the OR-connector in the process. From this paper it will become clear that you either have both events

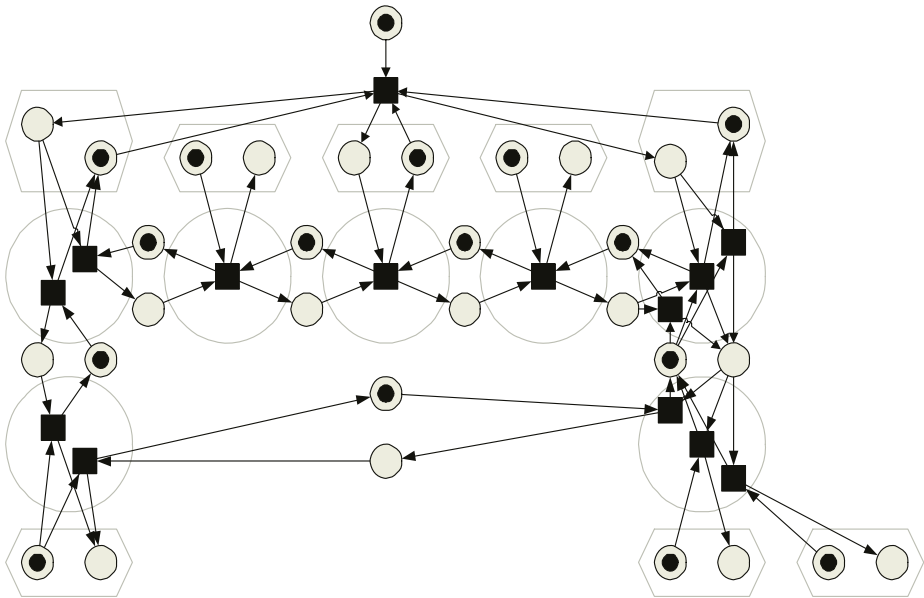


Fig. 9. Petri net translation of the reduced EPC

“Ready for analysis” and “Allowed final markings”, or you have “All allowed OR transitions are removed”. However, from the description of the EPC, this is not clear. Therefore, we require the user to select those possible outcomes that correspond to correct executions of the process.

**The Decision Process.** Finally, we have all the ingredients we need to decide whether the EPC is correct. We have a state space, of which we know all initial states and all allowed final states. The first step toward the final decision is to color everything from the state space that appears on a path from a initial state to one of the allowed final states. The colored part of the state space then describes all the behavior that the user allows. Then, we look for all transitions that do not have a colored edge in the reduced state space. We call those transitions “not covered”.

In principle, transitions that are not covered show that there is possible incorrect behavior. Translating this back to an EPC would result in saying that a certain connector is used incorrectly. This is indeed the case for connectors of type XOR and AND. However, for connectors of type OR, we need to perform an additional step. When people use connectors of type OR, they do not necessarily want all the possible behavior to appear. For example an OR split on two functions  $A$  and  $B$  can be used to express that you want to execute either  $A$ , or  $A$  and  $B$ , but never just  $B$ . In the verification process, this needs to be taken into account. If for example the transition that goes only to  $B$  is not covered, then it can safely be removed. However, this can only be done if the transition

to  $A$  and  $B$  is covered. This check is performed for all transitions that belong to OR connectors and are not covered. Some of them will be removed from the Petri net. The state space is then recalculated without the need for user interaction. Again, the coloring process is repeated and finally, when we know all the transitions that are covered, we can provide the final answer.

There are three possible answers, namely:

**The EPC is correct.** This is the case if the entire state space is colored. If the EPC is correct, then it is always possible to execute the process without ending up in some undesired state.

**The EPC can be correct.** This is the case if the state space is not entirely colored, but all transitions are covered. This result tells the designer that the EPC can be executed, but special care has to be taken to make sure that an execution does not end up in some undesired result.

**The EPC is incorrect.** This is the case when not all transitions are covered. Basically this means that there is some part of the EPC that cannot be executed without running into some undesired behavior.

In this section, we have presented a step by step algorithm for the verification of EPCs. We have shown that we need user interaction on two levels, and that the resulting answer is not “black or white”. Instead, there is a gray area where the EPC can be executed correctly, but can also run into problems. This gray area is not a flaw of the verification process. Instead, it shows the difference between a conceptual modeling language such as EPCs and an executable specification in terms of a Petri net. The EPC should be used to talk about the process and not as an executable specification. However, it is possible to derive such an executable specification from the EPC.

## 6 Two Case Studies

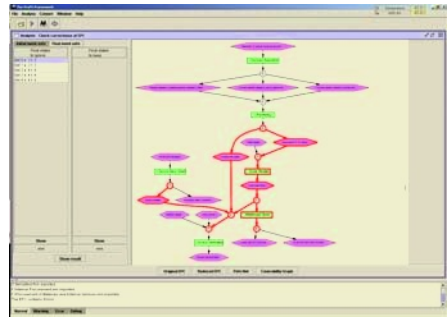
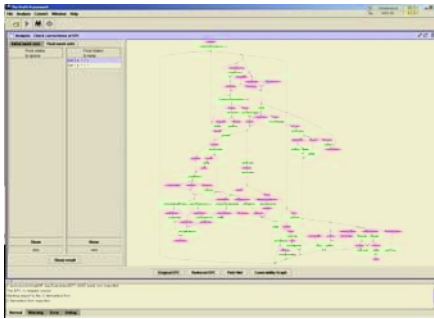
When developing methods for the verification of informal modeling languages, such as EPCs, there is a need to show applicability in real life. Therefore, we tested our approach in two different settings. The first case study was conducted within the Trade Department of a large Dutch bank.<sup>3</sup> There we applied our approach and tool on a trade execution process. We were not primarily interested in the outcome of the algorithm, i.e., whether the EPC analyzed was correct or not, but whether the consultants that modeled the EPC would understand the concepts described in this paper, and whether they would be able to use the tools we developed. The second case study was not conducted within an external organization. Instead we used our tool for the verification of some SAP reference models present in SAP R/3 and Aris for MySAP. Also in the second case study, we found some interesting problems.

---

<sup>3</sup> We cannot disclose the name of the bank.

## 6.1 Verification of Trade Execution Process in a Dutch Bank

Within the bank, business consultants made large EPCs modeling the trade execution process. They used the Aris toolset for modeling their business processes. They approached us to verify these processes. We applied the approach and the conclusion of the ProM tool was that the EPCs were correct. In other words, there existed possible executions that were not desirable. Using our tool, the consultants were able to identify the problem areas and from that they concluded that the model was correct and that the intended behavior would not lead to undesirable outcomes. Performing this test on their trade execution process made them decide to keep on using the ProM tool in the future. Figure 10 shows the trade execution process in our ProM tool.



**Fig. 10.** ProM showing the trade process **Fig. 11.** ProM showing the SAP process

## 6.2 Verification of SAP Reference Models

The SAP reference models are widely used in industry as a starting point for the configuration of SAP implementations. Of course, one would expect all these reference models to be correct, or at least to be possibly correct. Surprisingly, many reference models contained unrecoverable errors, and they would (if applied directly in industry) definitely lead to undesired behavior of the SAP system. In Figure 11 we show our tool highlighting the problem area of one of the SAP reference models. The model shown here is the “Procurement of Materials and External Services” process, where a mistake was made in one of the connectors, since it was modelled as a XOR-join instead of an AND-join.

The two case studies highlight the applicability of the approach. Unfortunately, we cannot elaborate on them because a detailed discussion would make the paper too long.

## 7 Conclusion

In this paper, we have presented an algorithm for the verification of EPCs. In contrast to many authors, we do not assume EPCs to be an executable specifica-

tion of a process, nor do we translate the EPC into one. In order to still be able to say something about the correctness of EPCs, we developed an interactive way of verifying EPCs. In this interactive process, we assume the user to have deeper knowledge of the EPC and we assume the user to be able to interpret the results. Besides that, we acknowledge the fact that EPCs are conceptual models and therefore our result cannot be expressed in a binary way. An EPC obviously is incorrect, if some part of it will always lead to undesired behavior, and it is correct if no part will ever lead to undesired behavior. However, there is a gray area in between those two extremes, where the EPC does allow for undesired behavior on the level of the model. This however, does not mean that there is no way of deriving an executable specification using the EPC as a basis.

## References

1. W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.
2. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer-Verlag, Berlin, 2000.
3. W.M.P. van der Aalst, J. Desel, and E. Kindler. On the Semantics of EPCs: A Vicious Circle. In M. Nüttgens and F.J. Rump, editors, *Proceedings of the EPK 2002: Business Process Management using EPCs*, pages 71–80, Trier, Germany, November 2002. Gesellschaft für Informatik, Bonn.
4. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
5. W.M.P. van der Aalst, A. Hirsenschall, and H.M.W. Verbeek. An Alternative Way to Analyze Workflow Graphs. In A. Banks-Pidduck, J. Mylopoulos, C.C. Woo, and M.T. Ozsu, editors, *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE'02)*, volume 2348 of *Lecture Notes in Computer Science*, pages 535–552. Springer-Verlag, Berlin, 2002.
6. W.M.P. van der Aalst and A.H.M. ter Hofstede. Verification of Workflow Task Structures: A Petri-net-based Approach. *Information Systems*, 25(1):43–69, 2000.
7. J. Dehnert and W.M.P. van der Aalst. Bridging the Gap Between Business Models and Workflow Specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.
8. J. Dehnert and P. Rittgen. Relaxed Soundness of Business Processes. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068 of *Lecture Notes in Computer Science*, pages 157–170. Springer-Verlag, Berlin, 2001.
9. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
10. K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 335–354. Springer-Verlag, Berlin, 2003.

11. G. Keller, M. Nüttgens, and A.W. Scheer. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.
12. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading MA, 1998.
13. E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In J. Desel, B. Pernici, and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97. Springer-Verlag, Berlin, 2004.
14. P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event driven Process Chains. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998*, volume 1420 of *Lecture Notes in Computer Science*, pages 286–305. Springer-Verlag, Berlin, 1998.
15. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
16. H. Lin, Z. Zhao, H. Li, and Z. Chen. A Novel Graph Reduction Algorithm to Identify Structural Conflicts. In *Proceedings of the Thirty-Fourth Annual Hawaii International Conference on System Science (HICSS-35)*. IEEE Computer Society Press, 2002.
17. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
18. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
19. W. Sadiq and M.E. Orlowska. Modeling and verification of workflow graphs. Technical Report No. 386, Department of Computer Science, The University of Queensland, Australia, 1996.
20. W. Sadiq and M.E. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In M. Jarke and A. Oberweis, editors, *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE '99)*, volume 1626 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, Berlin, 1999.
21. W. Sadiq and M.E. Orlowska. Analyzing Process Models using Graph Reduction Techniques. *Information Systems*, 25(2):117–134, 2000.
22. A.W. Scheer. *Business Process Engineering, Reference Models for Industrial Enterprises*. Springer-Verlag, Berlin, 1994.
23. H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 475–484. Springer-Verlag, Berlin, 2000.

# Measurement Practices for Knowledge Management: An Option Perspective

An-Pin Chen and Mu-Yen Chen

Institute of Information Management, National Chiao Tung University,  
Hsinchu, 300, Taiwan, ROC

{apc, mychen}@iim.nctu.edu.tw

**Abstract.** This article develops an option pricing model to evaluate knowledge management (KM) activities from the following perspectives: knowledge creation, knowledge conversion, knowledge circulation, and knowledge carry out. This paper makes three important contributions: (1) it provides a formal theoretical grounding for the validity of the Black-Scholes model that might be employed to KM; (2) it proposes a measurement framework to enable leveraging knowledge assets effectively and efficiently; (3) it presents the first application of the Black-Scholes model that uses a real world business situation involving KM as its test bed. The results prove the option pricing model can be act as a measurement guideline to the whole KM activities.

## 1 Introduction

In a knowledge economy where the only certainty is uncertainty, one source of lasting competitive advantage is knowledge and its manipulation [26]. Today, there is a growing recognition by researchers and practitioners about the importance of managing knowledge as a critical source for competitive advantage. As the resource commitments to knowledge management (KM) continue to escalate, the following types of questions are being asked more frequently than ever before: Is that investment in KM really worthwhile? Is that KM we implemented a success? Is our KM productive and effective?

Recent surveys indicate that issues such as ‘measuring the value of KM’ and ‘evaluating KM performance’ are of great importance to managers in places like Asia [1], the United States [29] and the United Kingdom [30]. Given the increasing role of KM in upgrading business competitiveness, the wide interest of managers in measuring and evaluating both KM performance and benefit is not surprising [8].

At another level of analysis, the productivity paradox has become a contentious issue among both economists and the information technology (IT) community [7] [28]. Indeed, many KM practitioners have used IT to practice KM in form of a knowledge management system (KMS). Unfortunately, several researches indicate that while the level of IT investment is correlated to corporate revenues, it is not correlated to either productivity or profitability [11] [15]. Managers have found it difficult to demonstrate tangible returns on the resources expended to plan, develop, implement and operate KM. For example, since effectiveness and innovation cannot be readily quantified in

terms of traditional outputs, these improvements are not reflected in economic efficiency statistics. Certainly, the fundamental issue of measuring and evaluating KM investments and performance remains unresolved.

This paper aims at proposing option pricing models in such a way that they become part of the managerial practice when evaluating KM solution. Its main contribution is the description of a real-life case study that demonstrates the usage of option valuation methods for analyzing KM. Regarding the organization of this paper; we start by giving an overview of prior research on KM evaluation in Section 2. We then describe the methodology of KM evaluation and integrate with balanced scorecard (BSC) in Section 3. Section 4 briefs on how the option models can serve as an evaluation tools for KM managers. The above-mentioned case study is presented in section 5. Finally, the conclusion and future work are discussed in Section 6.

## 2 Preliminary

KPMG [22] reports that the reasons for the creation of knowledge management initiatives cited by most companies are facilitating better decision making, increasing profit, and reducing costs. However, KM suffers from the same challenges as many other management issues: it assumes that knowledge is a ‘thing’ which is amenable to being ‘managed’ by a ‘manager’. First, which KM process is the key point to achieve competitive advantage? Second, which measurement method is the remarkable view-point to appraise KM performance?

KM performance measurement methods are broad category of research issues. We can see the method developments are diversified due to researchers’ backgrounds, expertise, and problem domains [23]. In our research, we can classify KM evaluation methods according to three types: qualitative and quantitative, financial and non-financial, internal and external performance approaches.

### 2.1 Qualitative and Quantitative Approach

A qualitative research approach was refined using the outcomes of a pilot study and reviews by researchers of organization learning. For example, the success of knowledge sharing in organizations culture, are not only technological but also related to behavior factors. Besides, expert interviews, critical success factors method (CSFs), and questionnaires are used to implement qualitative methods for exploring specific human problem.

**Table 1.** The benefits in the qualitative and quantitative index

Knowledge Management Benefits	
Qualitative Index	Quantitative Index
<ul style="list-style-type: none"> <li>● Improving employees skills</li> <li>● Improving strategies quality</li> <li>● Improving core business processes</li> <li>● Developing customer relationship</li> <li>● Developing supplier relationship</li> <li>● Developing innovative cultures</li> </ul>	<ul style="list-style-type: none"> <li>● Decreasing operation costs</li> <li>● Decreasing product cycle time</li> <li>● Increasing operation productivity</li> <li>● Increasing market sharing</li> <li>● Increasing shareholder equities</li> <li>● Increasing patent income</li> </ul>



In opposition, a quantitative research approach was designed to represent a tangible, visible and comparable 'ratio'. It can be measured by financial and non-financial index. We will discuss in next paragraph. Table 1 shows the KM benefits and classes with qualitative or quantitative indexes.

## 2.2 Financial and Non-financial Approach

Traditional quantitative methods focus on well-known financial measures, such as the payback period, the return on investment (ROI), the net present value (NPV), the return of knowledge (ROK), and the Tobin's  $q$ . These methods are best-suited to measure the value of daily transaction processing systems. Unfortunately, evaluation methods that rely on financial measures are not as well-suited for complicated IT applications. These systems typically seek to provide a wide range of benefits, including many that are intangible in nature. For example, it is difficult to quantify the full value of a point-of-sales (POS) system [4] or an enterprise resource planning (ERP) system [32].

Non-financial measures method is different from traditional financial statement analysis. It uses non-financial index, such as the frequencies, times, counts, and numbers. For example, the topic numbers of discuss board in KMS, are related to behavior factors and system usage situation.

## 2.3 Internal and External Performance Approach

Internal performance measurement methods focus on process efficiency and goal achievement efficiency. These methods evaluate KM performance through the gap between target and current value. The well-known methods are including ROI, NPV, balanced scored (BSC), and activity-based costing (ABC).

External performance measurement methods always compare itself with benchmark companies, primary competitions, or whole industry average. For example, benchmarking is the process of determining who is the very best, who sets the standard, and what that standard is. When we apply the benchmarking concept in business, the following types of questions are being asked: Which company has the best manufacturing operation? And how do we quantify that standard?

## 2.4 Option Valuation Approach

A number of researchers have written on the use of option models in IT investment decision making. The pioneering work of DosSantos [13] employs Margrabe's exchange option model [25] for valuing an IS project that uses a novel technology for testing. He argues the option model would be better than NPV to evaluate new IT project. Similarly, Kambil et al. [17] use the Cox-Rubinstein binomial option pricing model [10] to determine whether or not a pilot project should be undertaken.

For a software platform, several options usually are relevant. In analogy to Kester's "growth options" for firms [21], Taudes investigates options for evaluating "software growth option" [31], which can be valued software platforms and benefit.

Benaroch and Kauffman [4] investigate the problem of investment timing using the Black-Scholes model in a real world case study dealing with the development of point-of-sale (POS) debit service. Their contributions are not whether an investment

should be undertaken, but when to exercise the option held, i.e., when to implement a particular IT solution. In a follow-up paper, Benaroch and Kauffman [5] use sensitivity analysis to probe Black-Scholes valuation for IT investment opportunities. Taudes et al. [32] also compare NPV with Black-Scholes valuation method for employing SAP R/2 or to switch to SAP R/3. These results also indicated that, in the absence of formal evaluation of the time option, traditional approaches for evaluating information technology investments would have produced wrong recommendations.

### 3 Method and Evaluation Design

A universally accepted definition of KM does not yet exist. While there is debate as to whether knowledge itself is a cognitive state, a process, an object, the description of KM as a process, based on understanding organization as a knowledge system [14]. This view examines the nature of individual knowledge and collective knowledge, and their interactions.

#### 3.1 The Methodology of KM Evaluation

While authors differ in the terminology used in describing the KM process, the aggregate of their works can be described as a simple KM process as depicted in Fig.1. We generalized a conclusion from a collection of related KM researches and defined the “4C” process of KM activities: creation, conversion, circulation, and carry out.

Knowledge creation relates to knowledge addition and the correction of existing knowledge. Nonaka and Takeuchi [27] suggest four modes of knowledge creation: socialization, externalization, internalization, and combination. The model emphasizes interactions between individuals and organizations.

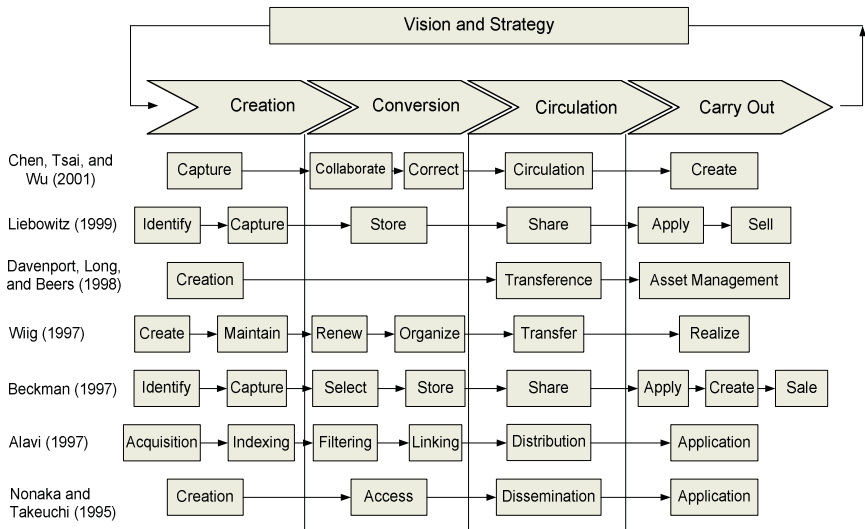


Fig. 1. KM process (see [2] [3] [9] [12] [24] [27] [33])

Knowledge conversion relates to individual and organizational memory. While organizational memory reflects the shared interpretation of social interactions, individual memory depends on the individual's experiences and observations.

Knowledge circulation is the dyadic exchange of knowledge between source and receiver. Transfer occurs at various levels: Transfer of knowledge between individuals, from individuals to explicit sources, from individuals to groups, between groups, across groups, and from the groups to the organization.

An import aspect of the knowledge carry out is that the source of competitive advantage resides in the knowledge itself. Here a major challenge is how to integrate internal knowledge and the knowledge gained from outside.

In order to present important research issues the pursuit of which would lead to the enhancement of knowledge usage in an organization, research questions related to each step of KM process can be integrated into four perspectives with BSC framework.

### 3.2 The Integration with BSC Framework

Underlying Kaplan and Norton's [18,19,20] concept of the BSC is that all aspects of measurement have their drawbacks; however, if companies offset some of the drawbacks of some measures with the advantages of others, the net measure can lead to decisions resulting in both short term profitability and long term success. As a result, they suggest that financial measures be supplemented with additional ones that reflect customer satisfaction, internal business processes, and the ability to learn and grow.

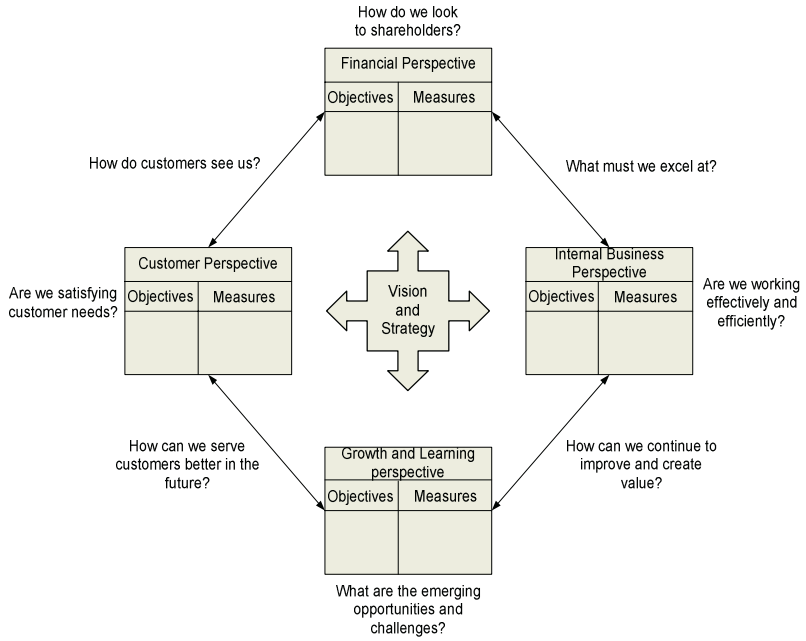
In a BSC framework, there are some metrics that drives performance improvement and enables the top management team to make well-informed decisions that prepare their organization for the future. The major elements are including: (1) vision: an image of what the organization will look like and do in the future; (2) strategy: that gives a sense of purpose to their organization; (3) objectives: the mission and vision are translated into objectives; (4) performance measures: the objectives can be measured through well-chosen indicators. Table 2 outlines the four perspectives included in a balanced scorecard, and Fig.2 shows the relationships between them.

The BSC concept can also be applied to measure, evaluate and guide activities that take place in specific functional areas of a business. For this reason, we integrated the conception of BSC and 4C process of KM.

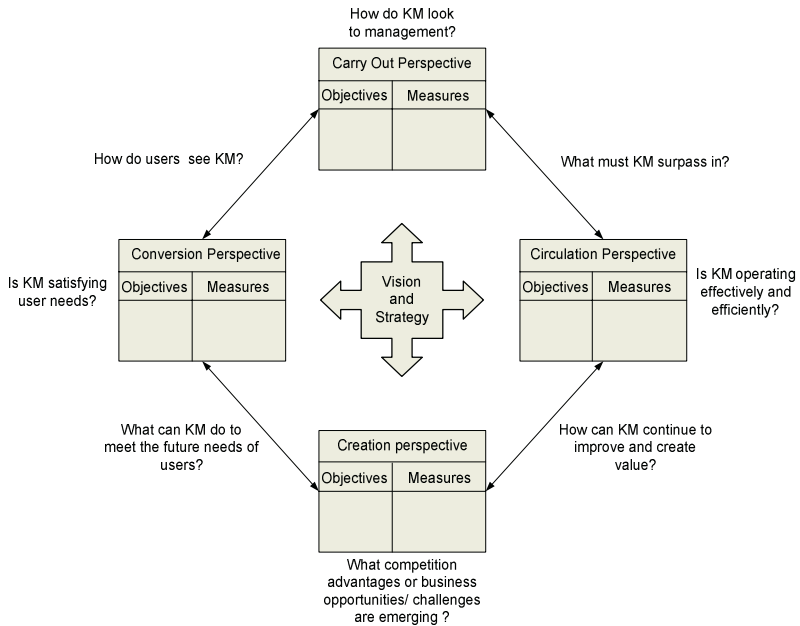
The following four perspectives have been suggested for a balanced KM scorecard: creation, conversion, circulation, and carry out. The relationship between these four new perspectives is illustrated in Fig.3.

**Table 2.** The four perspectives in a BSC

Perspective	Mission
Financial	To focus on the themes of (1) revenue growth and mix, (2) cost reduction and profitability, and (3) asset utilization and investment strategy.
Customer	To achieve desired overall performance, by improving customers' satisfaction, retention, and acquisition rate.
Internal Business	To identify processes those are most critical for achieving customer and ownership objectives, i.e., quality, cycle time, and innovation.
Growth and Learning	To identify needed developments within the organization to provide the infrastructure for future growth, i.e., employee capabilities, productivity, and empowerment.



**Fig. 2.** Relationship between the four perspectives in the BSC



**Fig. 3.** Relationship between the four perspectives in the balanced KM scorecard

## 4 Applying the Black-Scholes Model

The field of finance has developed a variety of option pricing models, with the fundamental ones being the Black-Scholes model. Because these models were originally developed to evaluate options on securities traded in the financial markets, they make certain assumptions that more naturally apply to options on traded assets. Over time, these models and their extensions have also been used in a variety of evaluative settings involving capital budgeting investments embedding real option. This paper makes three important contributions in this context: (1) it provides a formal theoretical grounding for the validity of the Black-Scholes model that might be employed to KM; (2) it proposes a measurement framework to enable leveraging knowledge assets effectively and efficiently; (3) it presents the first application of the Black-Scholes model that uses a real world business situation involving KM as its test bed.

### 4.1 Fundamental Option Pricing Model

In section 2, we sought a range of issues for KM valuation. Consequently, the key to understanding the KM performance evaluation in which option pricing is worthwhile to use relates to basic elements of the Black-Scholes model. For example:

(1) KM infrastructure investments often are made without any immediate expectation of payback. However, these can be converted investment opportunities into the option's underlying asset. Some examples of these investments include intranet and Internet environment, data warehousing and data mining technologies, and web service.

(2) KM embedded technologies are often difficult to forecast value payoffs in the face of unpredictable, implementation, and maintenance costs. Some examples of these technologies include search engine, enterprise information portal, and automated workflow systems.

(3) Knowledge investments represent that the knowledge is a core part of a company's competition advantages. Therefore, knowledge can be viewed as a product and gain tangible or intangible profits. Nevertheless, the knowledge has its "product life cycle" through newborn, mature, and abandoned phase. Here, the knowledge usage conception is similar to option pricing as time remaining to exercise.

### 4.2 Assumption About Black-Scholes Model

The Black-Scholes option pricing formula [6] prices European call or put options on a stock that does not pay a dividend or make other distributions. The formula assumes the underlying stock price follows a geometric Brownian motion with constant volatility.

(1) The Definition of Black-Scholes formula

$$\text{Option pricing formula prices} = \text{Intrinsic Value} + \text{Time Value} \quad (1)$$

Equation (1) can be explained that perfect financial markets are arbitrage-free in the sense that no investor can make a profit without taking some risk or expending some capital. Such gains could be made if an option were priced differently than a portfolio consisting of the underlying asset and a risk-less security with the amounts

being continuously adjusted so that the value of the portfolio replicates the value of the option. In equation (1), the value of a company or an asset based on an underlying perception of the value is called intrinsic value. For call options, this is the difference between the underlying stock price and the strike price; and further, time value is represented the portion of the option premium that is attributable to the amount of time remaining until the expiration of the option contract. Basically, time value is the value the option has in addition to its intrinsic value.

(2) Applying the Black-Scholes formula

In the Black-Scholes model [16], the value of a call option is its discounted expected terminal value,  $E [C_T]$ . The current value of a call option is given by  $C = E [C_T] (1 + r)^{-t}$ , where  $(1 + r)^{-t}$  is the present value factor for risk-neutral investors. A risk-neutral investors is indifferent between an investment with a certain rate of return and an investment with an uncertain rate if return whose expected vale matched that of the investment with the certain rate of return. Given that  $C_T = \max [0, S_T - K]$ , and assuming that  $S_T$  is log-normally distributed, it can be shown that:

$$\text{Black-Scholes formula: } C = S \cdot \Phi(d_1) - K (1 + r)^{-t} \Phi(d_2) \tag{2}$$

where

$$d_1 = \frac{\ln \frac{S}{K} + (r + 0.5 \sigma^2) t}{\sigma \sqrt{t}}$$

$$d_2 = \frac{\ln \frac{S}{K} + (r - 0.5 \sigma^2) t}{\sigma \sqrt{t}} = d_1 - \sigma \sqrt{t}$$

As shown in Equation (2), the Black-Scholes formula contains fewer parameters that are easier to determine. In addition to “ease to use” issue, applying option pricing concepts is attractive because of the conceptual clarity it brings to the analysis. Many knowledge management initiatives indicate the high potential variance of expected revenues from KM would be the key element in making the right decision. In this sense, option pricing seemed just right. We assume parameters of Black-Scholes model to be applied for KM. We employ the following notation in Table 3.

**Table 3.** The notations for Black-Scholes option pricing model and apply to KM

Notation	Black-Scholes Option Pricing Model	Apply to KM
C	The theoretical call premium	Value of investment
S	The value of option’s underlying stock price	Value of expected revenues
K	The option’s exercise price	Actual costs / expenses
σ	The standard deviation of the expected rate of return on S	Uncertain factors
Φ(d <sub>1</sub> )	The exposure of the option price with respect to the stock price	Measurement of KM investment and output
Φ(d <sub>2</sub> )	The cumulative standard normal distribution evaluated at (S>K) or (S<K)	Probability of KM success or fail

## 5 Case Study

In this paper, we use case study methodology to evaluate the performance of option pricing model, quite a lot of tests are preformed. To demonstrate how the test was executed, one experiment which is involves one high-technical company was selected and the research process as shown in Fig.4.

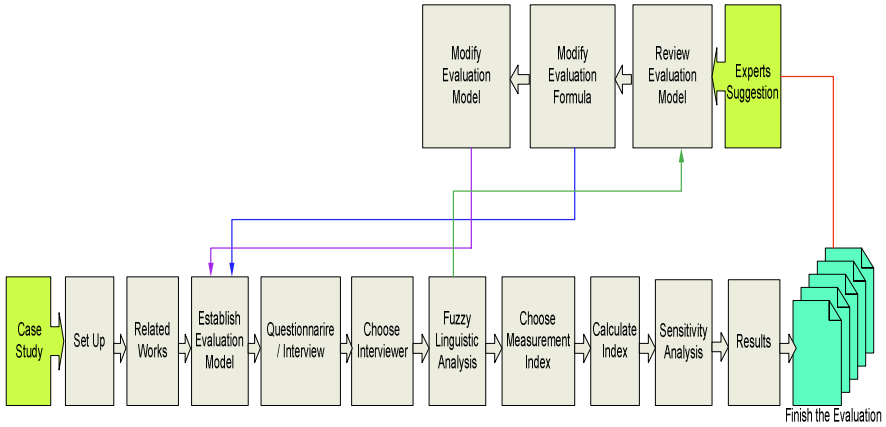


Fig. 4. Case study research process

### 5.1 General KM Evaluation

In order to acquire the importance for each measure in proposed four perspectives balanced KM scorecard, we design a questionnaire and interview end-users. Additional, we use fuzzy linguistic analysis to decide the default value for importance in each measure.

(1) The questionnaire analysis

In this questionnaire survey, 74 questionnaires were sent, 60 valid questionnaires were retrieved, and the ratio of valid retrieves was 81%.

(2) Setting up triangular fuzzy number

We used fuzzy Delphi method to adjust the fuzzy weight value for each measure. Then, we calculated the triangular fuzzy number for each measure. For example, the final measure of “innovation ability” in knowledge creation perspective was described as follows.

$$\tilde{W}_{innovation} = (0.2, 0.61, 0.9)$$

$$a_{innovation} = 0.2$$

$$b_{innovation} = \frac{1}{60} [(14 \times 0.8) + (9 \times 0.3) + (11 \times 0.75) + (22 \times 0.5) + (4 \times 0.9)] = 0.6125$$

$$c_{innovation} = 0.9$$

Where  $\tilde{w} = (a_k, b_k, c_k), k = 1, 2, \dots, n$  (3)

$$a_k = \text{Min}_i \{a_{ik}\}, \quad b_k = \frac{1}{m} \sum_{i=1}^m b_{ik}, \quad c_k = \text{Max}_i \{c_{ik}\}$$

As shown in Equation (3),  $\tilde{w}$  represents the  $k^{\text{th}}$  measure’s importance of the  $l^{\text{th}}$  participation’s valuation.

(3) Averaging the evaluation measures

After above step, we average each fuzzy weight value  $\tilde{W}_k$ , and get the mean value  $S_k$ . For example, the mean value  $S_k$  of “innovation ability” in knowledge creation perspective was described as follows. Table 4 shows an example of four perspectives measures value.

$$S_{\text{innovation}} = \frac{0.2 + 0.61 + 0.9}{3} = 0.57$$

According to the results, we can understand the KM performance in each perspective. However, we cannot gather significant discoveries because of there are not differentiable measure values in Table 4. Therefore, we will use our proposed option pricing model to estimate KM performance for each perspective.

**5.2 Using Black-Scholes Option Pricing Model**

In this section, we use Black-Scholes model to estimate knowledge creation perspective as an example. In Equation (4) ~ (6), we use parameters of Black-Scholes model to calculate the appropriate value which can be represented total key performance index (KPI). Even as Equation (7), we can determine which KM process or perspective must to be improved by KPI. As shown in Table 5, the knowledge carry out process is the most weakness in whole KM activities (PKI=0.0014). Therefore, the manager will enhance related objectives in this perspective according to the above statement.

(1) Calculating the investment costs of KM ( $S$ )

$$S_{KM} = \sum_1^l C_{KM} = \left( \text{equipment cost} + \text{labor cost} + \text{time cost} + \text{operation cost} \right) \quad (4)$$

(2) Calculating the expected revenues of KM ( $K$ )

$$K_{KM} = \sum_1^l R_{KM} = \left( \text{physical revenues} + \text{invisible revenues} \right) \quad (5)$$

(3) Calculating the uncertain factors ( $\sigma$ )

$$\sqrt{\sum_{i=1}^n \frac{(S_i - \bar{S}_{KM})^2}{n}}, \quad S_i = S_{KM}(t) - S_{KM}(t-1) \quad (6)$$

(4) Calculating the total key performance index (KPI)

$$BS_{KM} \text{ value} = KPI \quad (7)$$

$$KPI = C = S \cdot \Phi(d_1) - K(1+r)^{-t} \Phi(d_2)$$



**Table 4.** The value of four perspectives in a balanced KM scorecard

KM Process	Objective	Measure	Value
Creation	Continuously training and development	0.64	0.65
	The innovation abilities for users	0.57	
	The average seniority for users	0.73	
Conversion	Users' experiences	0.62	0.54
	Users' professional skills	0.48	
	User's satisfaction	0.52	
Circulation	The incentive systems for users	0.55	0.54
	The sharing culture among users	0.57	
	The standardization of documents	0.51	
Carry Out	Ensure the KM project provides business value	0.62	0.51
	The quantities / qualities of knowledge database	0.42	
	The numbers of patents	0.50	

**Table 5.** KPI of Black-Scholes model

KM Process	Intrinsic value (S-K)		Time value		Black-Scholes option value
	S	K	$\sigma$	t	
Creation	2400	3215	12%	2	35.8174
Conversion	2150	3000	12%	2	20.2288
Circulation	2000	3400	12%	2	1.0652
Carry out	2000	4600	12%	2	<b>0.0014</b>

**5.3 Sensitivity Analysis Using Black-Scholes Model Derivatives**

Sensitivity Analysis aims at showing how the results of an analysis change as its underlying assumptions change. As shown in Table 6, we can evaluate the benefits or costs in the KM project with derivative analysis.

**Table 6.** Sensitivity Analysis

KM Process	(Delta)	(Gamma)	(Vega)	(Theta)	(Rho)
Creation	0.176	0.0006	878.0326	-49.5336	773.0867
Conversion	0.1208	0.0006	611.0055	-32.6966	478.8825
Circulation	0.0098	0.0001	73.9137	-3.3267	36.9776
Carry out	0.0	0.0	0.2634	-0.0093	0.0744

A major challenge for KM research lies in making models and theories to evaluation its performance and values. However, traditional methodologies have long relied on NPV, simple cost-benefit analysis, critical success factors and other less-structured techniques to perform their assessment. Thus, our experiment has been to critically

review the case for using option pricing as a basis for KM performance analysis and to evaluate its merits in an actual real word business setting.

## 6 Conclusions

In this paper, we have made the argument the option pricing model can be applied to KM performance valuation. In the initial stage, we generalized a conclusion from a collection of related KM researches and defined the 4C process of KM activities: creation, conversion, circulation, and carry out. In the next stage, we pursuit of which process would lead to the enhancement of KM performance in a firm, hence we integrated KM process into four interrelated main research streams with BSC framework. Finally, we illustrated how the Black-Scholes model can be applied in the case of a real world KM performance option, where significant uncertainties that are not appropriately handled using traditional financial analysis were present. The results have proven the option pricing model can be act as a measurement guideline to the whole KM activities.

Future research will focus on several issues. First, we will investigate other firms into KM performance valuation by our approach. Second, we will gauge the risk associated with the KM project in a firm. Finally, we will improve the estimation parameters methods. Especially, more general guidelines could make the option valuation of KM performance less time-consuming and more reliable.

## References

1. Ahn, J.H., and Chang, S.G.: Assessing the Contribution of Knowledge to Business Performance: the KP3 Methodology. *Decision Support Systems*, Vol. 36 (2004) 403 – 416
2. Alavi, M. KPMG Peat Marwick U.S.: One Giant Brain. Harvard Business School(Case), 9- 397- 108 (1997)
3. Beckman, T.: A Methodology for Knowledge Management. *Proceeding of the IASTED International Conference on AI and Soft Computing* (1997)
4. Benaroch M. and Kauffman R.J.: A Case for Using Real Options Pricing Analysis to Evaluate Information Technology Project Investments. *Information Systems Research*, Vol. 10, Iss. 1 (1999) 70-86
5. Benaroch M., and Kauffman R.J.: Justifying Electronic Banking Network Expansion Using Real Options Analysis. *MIS Quarterly*, Vol. 24, Iss. 2 (2000) 197-225
6. Black, F., and Scholes, M.: The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, Vol. 81, No. 3 (1973) 637-659
7. Brynjolfsson, E. and Hitt, L.: Paradox Lost? Firm-level Evidence on the Returns to Information Systems Spending. *Management Science*, Vol. 42, Iss. 4 (1996) 541-558
8. Brynjolfsson E., Renshaw, A.A., Alstyne, M.V.: The Matrix of Change. *Sloan Management Review*, Vol. 38, No.2 (1997) 37–54
9. Chen, M.Y., Tsai, M.J., and Wu, H.R.: The Research of KM Operation Module in Science & Technology Industry – Case Study of TSMC. *Proceedings of the 12th International Information Management Conference*, Taiwan (2001) A-75
10. Cox, J., Ross, S., and Rubinstein, M.: Option Pricing: A Simplified Approach. *Journal of Financial Economics*, Vol. 6 (1979) 229-263
11. Das, A.: Knowledge and Productivity in Technical Support Work. *Management Science*, Vol. 49, Iss. 4 (2003) 416-431

12. Davenport, T.H., Long, D.W., and Beers, M.C.: Successful Knowledge Management Projects. *Sloan Management Review*, Vol. 39, No. 2 (1998) 43-57
13. DosSantos, B.L.: Justifying Investments in New Information Technologies. *Journal of Management Information Systems*, Vol. 7, Iss. 4 (1991) 71-90
14. Grant, R.M.: Prospering in Dynamically-Competitive Environments: Organizational Capability as Knowledge integration. *Organization Science*, Vol. 7, No. 4 (1996) 375-387.
15. Holsapple, C.W., and Singh, M.: The Knowledge Chain Model: Activities for Competitiveness. *Expert Systems with Applications*, Vol. 20 (2001) 77-98
16. Hull, J.C.: *Options, Futures, and Other Derivative Securities* (2<sup>nd</sup> edition). Prentice Hall, Englewood Cliffs, NJ (1993)
17. Kambil, A., Henderson, J., and Mohsenzaden, H.: Strategic Management of Information Technology Investments: An Option Perspective. in *Strategic Information Technology Management: Perspectives on Organization Growth and Competitive Advantage*, Idea Publishing Group (1993) 161-178
18. Kaplan R., Norton, D.: The Balanced Scorecard: Measures that Drive Performance. *Harvard Business Review*, Vol. 70, No. 1 (1992) 71-79
19. Kaplan R., Norton, D.: Putting the Balanced Scorecard to Work. *Harvard Business Review*, Vol. 71, No. 5 (1993) 134-142
20. Kaplan R., Norton, D.: Using the Balanced Scorecard as a Strategic Management System. *Harvard Business Review*, Vol. 74, No. 1 (1996) 75-85
21. Kester, W.C.: Today's Options for Tomorrow's Growth. *Harvard Business Review*, Vol. 62 (1984) 153-161
22. KPMG.: Knowledge Management Research Report, (1998) URL: <http://www.kpmg.com>
23. Liao, S.H.: Knowledge Management Technologies and Applications—Literature Review from 1995 to 2002. *Expert Systems with Applications*, Vol. 25 (2003) 155-164
24. Liebowitz, J.: Key Ingredients to the Success of an Organization's Knowledge Management Strategy. *Knowledge and Process Management*, Vol. 6, No. 1 (1999) 37-40
25. Margrabe, W.: The Value of an Option to Exchange One Asset for Another. *Journal of Finance*, Vol. 33, No. 1 (1978) 177-186
26. Nonaka, I.: The Knowledge Creating Company. *Harvard Business Review*, Vol. 69, No. 6 (1991) 96-104
27. Nonaka, I and Takeuchi, H.: *The Knowledge Creating Company*. New York, NY: Oxford University Press (1995)
28. Pinsonneault, A. and Rivard, S.: Information Technology and the Nature of Managerial work: From the Productivity Paradox to the Icarus Paradox. *MIS Quarterly*, Vol. 22, Iss. 3 (1998) 287-311
29. Schultze, U. and Leidner, D.E.: Studying Knowledge Management in Information Systems Research: Discourses and Theoretical Assumptions. *MIS Quarterly*, Vol. 26, No. 3 (2002) 213-242
30. Shin, M., Holden, T., and Schmidt, R.A.: From Knowledge Theory to Management Practice: Towards an Integrated Approach. *Information Processing and Management*, Vol. 37 (2001) 335-355
31. Taudes, A.: Software Growth Options. *Journal of Management Information Systems*, Vol. 15, Iss. 1 (1998) 165-185
32. Taudes, A., Feurstein, M. and Mild, A.: Options Analysis of Software Platform Decisions: A Case Study. *MIS Quarterly*, Vol. 24, Iss. 2 (2000) 227-243
33. Wiig, K.: Knowledge Management: Where Did It Come From and Where Will It Go. *Expert Systems with Applications*, Vol. 13, No. 1 (1997) 1-14

# An Ontological Approach for Eliciting and Understanding Needs in e-Services\*

Ziv Baida<sup>1</sup>, Jaap Gordijn<sup>1</sup>, Hanne Sæle<sup>2</sup>, Hans Akkermans<sup>1</sup>,  
and Andrei Z. Morch<sup>2</sup>

<sup>1</sup> Free University, FEW/Business Informatics,  
De Boelelaan 1083a, 1081 HV Amsterdam, The Netherlands  
{ziv, gordijn, elly}@cs.vu.nl

<sup>2</sup> Dep. of Energy Systems, SINTEF Energy Research,  
7465 Trondheim, Norway  
{Hanne.Saele, azm}@sintef.no

**Abstract.** The lack of a good understanding of customer needs within e-service initiatives caused severe financial losses in the Norwegian energy sector, resulting in the failure of *e-service* initiatives offering packages of independent services. One of the causes was a poor elicitation and understanding of the e-services at hand. In this paper, we propose an ontologically founded approach (1) to describe customer needs, and the necessary e-services that satisfy such needs, and (2) to bundle elementary e-services into needs-satisfying e-service *bundles*. The ontology as well as the associated reasoning mechanisms are codified in RDFS to enable software support for need elicitation and service bundling. A case study from the Norwegian energy sector is used to demonstrate how we put our theory into practice.

## 1 Introduction

Today, e-Business still focuses on supporting the production, sale, and consumption of *physical* products. However, in real-life, many products are actually *digital*, in contrast to goods that you can drop onto the floor. Consequently, a new paradigm in e-Business is emerging: *digital products* [20].

E-services are a web-based version of traditional services, defined as business activities, deeds and performances of a mostly intangible nature [16, 18, 25, 17]. In the rest of this paper we refer to this definition when we use the term ‘e-service’. Note that e-services are not the same as web-services. E-services are *digital* services, provisioned over the Internet, whereas web-services are *physical* services. A paradigm to arrive at truly distributed computing (more information on the differences between e-services and web-services can be found in [6]).

---

\* This work has been partially supported by the European Commission, as project No. IST-2001-33144 OBELIX (Ontology-Based ELeCTronic Integration of complex products and value chains) and by the Dutch Ministry of Economic Affairs, as the FrUX project (Freeband User eXperience).

In e-Business scenarios, it is important that all participants have a common understanding of the goods and services offered and requested. For instance, many efficiency gains in supply chain management rely on information integration along this chain; to arrive at such an integration each party in the chain should have the same understanding of the good or service to be delivered. Moreover, since e-Business scenarios use software components extensively almost by definition, it is also important to have a common, or at least machine interpretable understanding of the services and goods. As an example, consider a software component that proposes a bundle of services or goods (potentially delivered by multiple suppliers) that as a whole satisfies a specific customer need. Such a bundling component should be able to reason about meaningful combinations of products<sup>1</sup>. This can only be done if the service and good descriptions are machine interpretable. Making a shared, formal conceptualization of, in this case, goods and services is the field of ontologies [10]. Existing product ontologies have a strong emphasis on physical goods (see e.g. [2, 1]) and not yet on services.

To address this shortcoming, we developed and tested an ontology to represent e-services [3], and implemented a prototype software tool. This ontology can be used, first to build a catalogue of e-services, and second to compose bundles of e-services, which as a bundle satisfy a customer need. Bundling is a well known economic principle in building service offerings [16]. Elementary services in a bundle can be offered by different enterprises that form a partnership. An ontological approach is really needed here; all businesses in such a partnership should have a shared and formal understanding of their elementary e-services to facilitate automated reasoning about commercial viable e-service bundles.

So far, our ontology has been stated in supplier-oriented terminology. Customers have to formulate their customer needs (or requirements) in their own terminology to arrive at meaningful bundles of e-services. Obviously, a customer wants to express his needs in his own terminology, which can be different from the terminology used by the supplier. This paper presents an extension to our ontology that allows a customer to do so.

This work is unique in a number of ways. First, it recognizes the commercial nature of services, in contrast to web-service ontological approaches, which are strong at facilitating distributed computing but poor in the commercial interpretation of 'service'. Second, our ontology is capable of doing automated bundling of elementary e-services into e-service bundles using a configuration problem solving method borrowed from AI. And now, our ontology can be used to derive, based on customer needs, bundled e-service offerings.

Our work is not limited only to e-services, but applies to any service offerings. Nevertheless, our work is of much greater importance for e-services, since they require automating the processes that may otherwise be performed in the minds of service personnel. Consequently, for e-services realization it is absolutely necessary that domain knowledge is conceptualized, formalized and made machine-interpretable. This is what we aim to achieve in our work.

---

<sup>1</sup> Both *services* and *goods* are subclasses of *product*.

This paper starts with an introduction of our case study in Section 2. Then we present a summary of earlier work on our e-services ontology (Section 3). In Section 4 the ontological extension to cover customer needs, wants, and demands is introduced. Then, in Sections 5 and 6, we show how this extension uses known theory from Requirements Engineering to come from customer needs to service-outcomes, to be produced by suppliers. Section 7 discusses existing and envisioned tool support. Finally, in Section 8 we present our conclusions and identify directions for future research.

## 2 Case Study: Service Bundles in the Norwegian Energy Sector

This paper uses a case study, based on a real-life e-service elicitation project we carried out in Norway. The study at hand is about electricity supply. Electricity is an anonymous product. Due to a fierce competition in generation and supply of electricity, the difference in electricity retail prices per kWh between suppliers is diminishing. Electricity fits ideally to definition of “perfect substitute”, since electricity which is sold from one supplier has exactly the same physical characteristics as electricity sold by another supplier, and also prices are very similar. Consequently, many suppliers are seeking for ways to differentiate their offerings from competitors, so that customers are able to distinguish the individual offerings from suppliers. One way to do so is to add complementary and additional services such as Internet access and home comfort management to the electricity supply offering. In our study for an electricity supplier in Trondheim, Norway, we analyzed possible service bundles that can be offered via the Web to customers who wish to buy electricity. Service bundles had to be designed so that (1) they are commercially interesting, and (2) they meet customers’ demands. We focus on the first question in [5]. The present article focuses on the second question.

## 3 Ontological Framework: A Service Ontology

On a high level of abstraction, our service ontology [3] embodies three interrelated top-level perspectives:

The *customer value* perspective captures knowledge about adding economic value from a customer viewpoint. It expresses customer needs, expectations and experiences, and is driven by a customer’s desire to buy a certain service of a certain, often vaguely defined quality, in return for a certain sacrifice (including price, but also intangible costs such as inconvenience costs and access time). As the service value perspective is the main contribution of this paper, we elaborate more on this perspective in Section 4.

The *service provider* perspective represents the supply-side viewpoint: it provides a hierarchy of service components (e.g. a core service and supplementary services) and outcomes, as they are actually delivered by the service provider in order to satisfy customers’ needs.

The service offering perspective encapsulates knowledge about putting the service offering into operation in terms of business processes. In contrast to the usual production process of physical goods, customers often take active part in the service production process.

Our earlier work focused on the service offering perspective (see [7] for a detailed discussion). For the current discussion it suffices to understand that the provisioning of a service requires a set of resources and results in the availability of a set of services. Very often the service outcomes reflect the customer benefits from a service, whereas the customer sacrifice is mapped into service inputs (e.g. payment). Service inputs and outcomes are referred to as resources. Hence a service is described by its resources: its requires inputs and its generated outcomes. Section 4 presents the service value perspective. Sections 5 and 6 describe how both perspectives can be related in order to support reasoning on needs-driven service bundling.

### 4 Service Value Perspective

The service offering perspective of our service ontology describes service elements including their input- and outcome-resources, as well as customer requirements in terms. The motivation for doing so is that the service offering perspective aims at configuring the various e-services elements of different suppliers in a more comprehensive bundle, and for doing so we need the actual service elements that can be provisioned by these suppliers.

Customers however do not articulate their needs in terms of supplier-oriented requirements but employ their own, subjective terminology for expressing demands. To deal with these demands, we extend our ontology with resources, services, and miscellaneous constructs. In brief, customers state their demands, which can (partly) be satisfied by a series of bundled resources, which in turn are provisioned by services (see Figure 1).

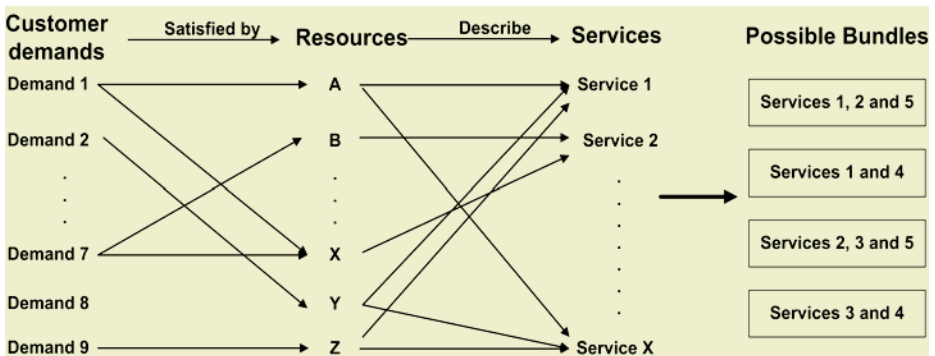


Fig. 1. Configuring service bundles based on customer demands

The service sub-ontology perspective is sketched in Figure 2 and is explained below.

**Needs, wants and demands.** The starting point for the discipline of marketing lies in the human needs and wants [18]. The term *need* refers to what humans need and want (to buy). Kotler [18] distinguishes between needs, wants and demands:

- A human *need* is a state of felt deprivation of some basic satisfaction.
- *Wants* are desires for specific satisfiers of these deeper needs.
- *Demands* are wants for specific products that are backed up by an ability and willingness to buy them.

Needs are often vague; the need for “financial security” can be interpreted in many ways. Customers concretize their needs by transforming them into wants and demands, for example based on exposure to existing services and to marketing campaigns. Often when a customer is interested in a service, he has already transformed his needs into wants and demands. He has, in fact, already found a solution for his problem (need). *Example*: indoor comfort (need); lighting (want); energy supply (demand). An exploration of customer needs, wants and demands for the energy utility we investigated is provided in Table 1.

**Service quality** is the degree and direction of the discrepancy between a customer’s expectations and the perception of the service [9]. Customer expect-

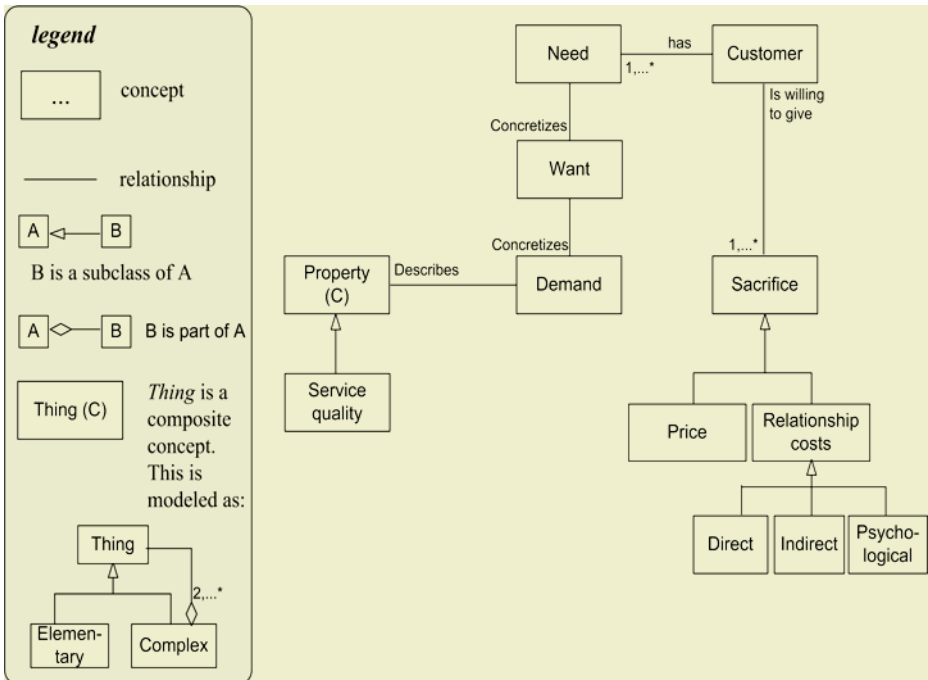


Fig. 2. Service sub-ontology representing the service (customer) value perspective



tations embrace several different elements, including desired service, predicted service and a zone of tolerance that falls between the desired and adequate service levels [8]. Expectations are based on word of mouth communications, personal needs, past experience and external communications from service providers [25]. At least two widely accepted generic methods for defining service quality are used in business science: that of the Nordic school [16] and that of the North American school (SERVQUAL, see [25]). *Example*: In electricity supply, quality can be seen as the allowed deviations in voltage, frequency, the allowed occurrences of outages, etc. Customers, for instance with respect to power outages, have different views on this (some customers have high-availability requirements, while others have not).

Next to quality, also other criteria may play a role, e.g. location and time (where and when the service should be provided). For this reason we introduced the concept **Property** in our ontology, as a super class of ‘service quality’. In the rest of this paper whenever we use the term ‘desired quality’ we refer also to other properties.

**Sacrifice.** The customer sacrifice includes the price of the service as well as relationship costs: direct (e.g. investment in office space), indirect (e.g. time that the customer has to devote to maintaining the relationship) or psychological costs (e.g. lack of trust in a service provider; unpleasant sensory experiences such as noise) [16]. *Example*: time spent waiting to be served; travel costs; switching costs (from one supplier to another).

## 5 From Service Value Perspective to Service Offering Perspective

### 5.1 Need, Want and Demand Hierarchies

The notions *need* and *service value* have been acknowledged in the field of Goal Oriented Requirements Engineering (GORE) as suitable for transforming high-level organizational needs to concrete system requirements [13]. We utilize this GORE terminology to relate the service-value perspective to the service-offering perspective.

Needs, Wants, and Demands (further collectively referred to as “needs”) capture the answer for the question why a service bundle is offered. Hence *need* are equivalent to *requirement* in system/software design: needs represent why a service bundle is needed; goals represent why a system/software is needed.

Goals, at different levels of abstraction, capture the various objectives that the system under consideration should achieve [22, 23]. Just like goal hierarchies exist in GORE [15], also the marketing literature acknowledges hierarchies of needs [18]. As shown in Section 4, a *need* is a state of felt deprivation of some basic satisfaction; it can be concretized by *want*, and further by *demand*: wants for specific products that are backed up by an ability and willingness to buy them.

Table 1 presents our hierarchy of needs, wants and demands (further referred to as ‘need hierarchy’) for the energy case study at hand. The notations H/I refer

**Table 1.** Customer needs, wants and demands for the energy utility TrønderEnergi

<i>Customer's Needs</i>	<i>Customer's Wants</i>	<i>Customer's Demands</i>
Indoor comfort (H,I)	Lighting (H,I); Home services (cooking, washing) (H); Comfort temperature (H,I)	Energy supply (H,I); Hot tap water (H,I); Room heating (H,I); Air conditioning (H,I)
	Energy regulation for budget control (H,I)	Energy regulation for budget control (H,I), with different characteristics (manual / automated; on-site regulation / location-independent)
	Temperature regulation for increased comfort (H,I)	Temperature regulation (H,I) with different characteristics (manual / automated, on-site regulation / location-independent)
Social contacts and Recreation (H); Business contacts (I)	Communication (H,I)	Telephone line (H,I); Mobile phone line (H,I); Internet (broadband) (H,I)
Safety (H,I)	Increased security (H,I); Reduced insurance premium (H)	Safety check of electrical installation (H); Internal control of electrical installation (I)
IT support for business (I)	IT-services (I)	ASP-services (I); Hardware (I); Software (I)

to the customer type: Household or Industrial. As can be seen from the table, demands may either indeed refer to concrete services (e.g. a mobile phone line), or be more abstract, when a customer does not necessarily know which service can satisfy his need, or when a diversity of solutions exists (e.g. the demand for temperature regulation).

### 5.2 A Need Goal Tree

Needs, wants, and demands can be arranged in an AND/OR goal tree as we know from GORE. In GORE, links between goals are aimed at capturing situations where goals positively or negatively support other goals [23]. Hard-goals can be refined through AND/OR graph structures. Soft-goal refinement uses the same AND/OR structures, as well as weaker links:  $\dots \vdash \dots$ ,  $\dots \dashv \dots$  and  $\dots \dashv \dots$ . By using links to refine goals it becomes possible

to reason about goals. We therefore suggest to introduce the same AND/OR refinements used in GORE also for need hierarchies. In such a need hierarchy, needs are concretized in wants, which in turn are concretized in demands. Demands are the leafs of the tree. An example of such a need-hierarchy is given in the left-most part of Figure 3. Note that the marketing literature does not make a distinction between types of needs or refinements in need hierarchies.

Our case study showed that the use of above refinement structures requires adding a *customer type* dimension, since customer needs differ per customer type, and thus the refinement changes per customer type. This needs-differentiation relates to the notion of *market segments* in marketing literature [18]: “a market segment is defined as a concept that breaks a market, consisting of actors, into segments that share common properties”. These common properties depend on the specific context.

For example, the customer want for ‘communication’ can be refined to three demands: (regular) telephone line, mobile phone line and Internet access. Whereas one customer may require only a regular phone line, another may want Internet access and a mobile phone line, and no regular phone line. This illustrates precisely why supplier stated requirements are not sufficient for e-service bundling: suppliers present services in terms of what they can offer, while customers initially think in vague terms such as ‘communication’.

Quality attributes are stated as properties of demands in Table 1. For instance, consider temperature regulation with quality properties *temperature range*, *temperature stability*, etc.

By following the AND/OR relations, the knowledge required for reasoning about potential service outcomes satisfying a need is thus available on the demand level, rather than on the more abstract want or need levels. Since demands are satisfied by outcomes (resources) provisioned by e-services, demands are a good starting point for finding service bundles that may satisfy higher needs.

## 6 Demands and Resources Are Features and Solutions

### 6.1 Demands Are Satisfied by a Service That Provides Certain Resources

The purpose of building a need hierarchy is twofold. First the hierarchy is used to find context depending demands, based on more abstract wants and needs. Second, found demands should be satisfied by service-outcomes (resources) provisioned by suppliers. We employ Feature-Solution graphs [11, 12] to relate demands and resources.

### 6.2 Linking Demands to Resources

A link between customer demands and resources provisioned by services can be viewed as a production system consisting of *resources*, a knowledge representation formalism used in the AI field. Production rules have the form: if

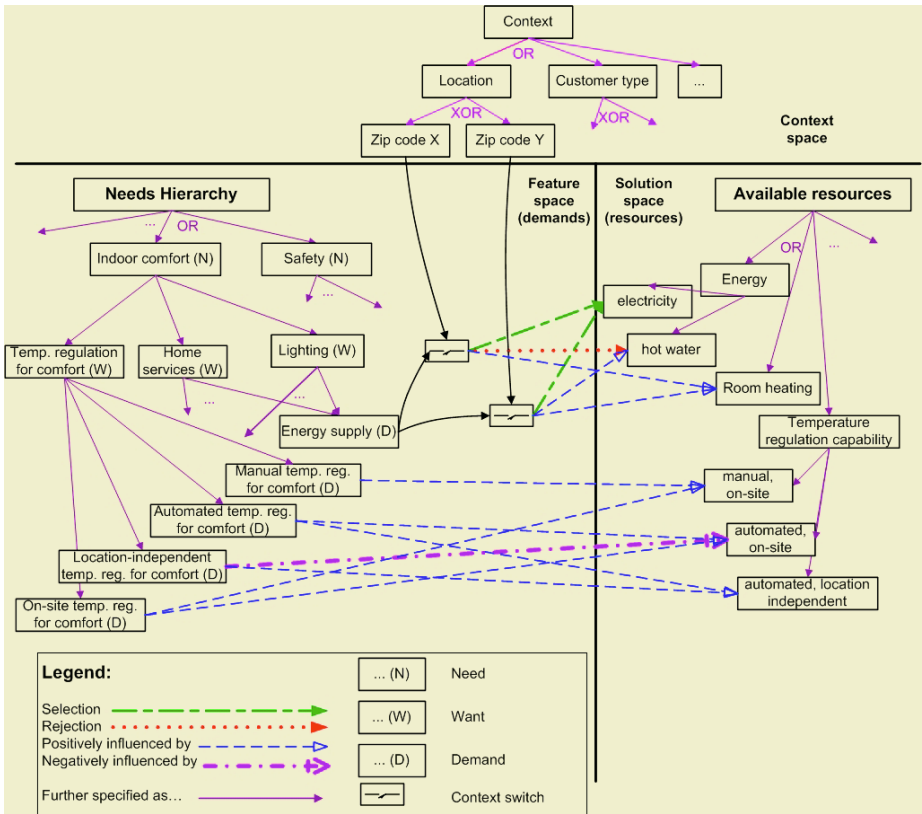


Fig. 3. Partial FS-graph of the energy case study

situation X is encountered then select solution Y. De Bruin et al. [11, 12] suggested the use of a context-aware partial FS-graph (FS-graph) to model these production rules. FS-graphs capture and document context-sensitive domain knowledge, so that it becomes possible to reason about feasible solutions and the requirements they support. An FS-graph includes three spaces, organized in hierarchies of AND-(EX)OR decompositions:

1. **Feature space:** describes the desired properties of the system (or: service) as expressed by the user. In our case, these are customer demands.
2. **Solution space:** contains the internal system (services) decomposition into resources that are required for or produced by available services.
3. **Context space:** contextual information relevant for the domain (e.g. customer types, geographic restrictions).

Links between elements of the Feature-space (demands) and elements of the Solution-space (resources) may have the semantics of  $A \rightarrow B$  (if demand A then resource B),  $A \rightarrow \neg B$  (if demand A then not resource B) or weaker relations:

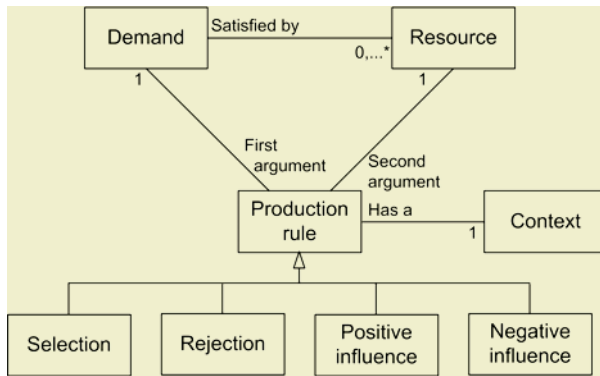


Fig. 4. FS-graph constructs in the service ontology

... or ... The FS-graph offers levels of flexibility as a result of the different decomposition possibilities of features and solutions.

An example FS-graph, adapted for our case, can be found in Figure 3. For visualization reasons we present only a fraction of the need hierarchy tree, and we mention the type of hierarchy (AND/OR/XOR) explicitly only in a few of the places. As can be seen, contextual information can change the behavior of a production rule. This is modeled by a ... If a switch node is selected, the switch is closed and establishes context-dependent relations between features and solutions [12]. Context may include location, but also customer type (see the discussion on market segments in Section 5.2).

While the FS-graph can be used to visually communicate above production rules, its constructs need to be added to the earlier presented service ontology in order to facilitate the automated support of linking demands and resources. Figure 4 shows how we incorporate FS-graph structures in the service ontology.

The service value perspective of our service ontology – including the concepts ... and ... – reflects a customer view on services. As such it is by definition context-sensitive: every customer type may have a different viewpoint on a service, based on his/her situation (time, location, role), on different expectations and on past experiences.

### 6.3 Relevance of Relations in the FS-Graph

We applied the FS-graph approach to the energy case study, considering customer demands as features, and available resources as solutions. This resulted in lessons learned regarding the four types of relations between features and solutions.

1. A ... relation hardly exists. There is not really a ... resource for a possible demand, but alternatives exist.

2. A *disqualify* relation may be required, but does not occur often. An example is the demand for energy supply with quality descriptor ‘green energy’ and the resource ‘energy’ with quality descriptor ‘nuclear’.
3. A *fulfill* relation is the basis for our model. It denotes that a customer demand can be fulfilled by certain resources, and hence by certain services.
4. A *disqualify* relation may occur, but not often. For example, when a demand is specified by quality descriptors implying that the customer is interested in a cheap service, whereas a suggested resource is specified by the quality descriptor ‘high’.

As can be seen, the *fulfill* relation plays the main role in our case. One could thus question the use of the FS-graph constructs, which include four relations. There are two answers for this question. First, a conceptualization and formalization of domain knowledge is an absolute necessity for automated reasoning and solving problems about that knowledge. The FS-graphs approach, using goal hierarchies, has shown to be an effective approach for making this knowledge explicit, and suitable for systematic automated reasoning using production rules. Second, when adding also acceptable sacrifices (i.e. price) to the graph, we will receive a richer FS-graph, in which the *disqualify* relation and the *fulfill* relation will occur much more often (a low acceptable sacrifice will disqualify expensive solutions).

### 6.4 Reflecting Back on the Case Study

We modeled a variety of services in the energy case study, including *energy supply*, *energy storage*, *energy distribution*, and *energy conversion*, and more (see [7]), analyzed links between services and customer demands, and created service bundles to satisfy customer demands. As a result of the modeling of service elements and the automated generation of service bundles, the energy utility at hand succeeded in defining service bundles for specific groups of customers in such a way that these bundles fit the requirements of their respective customers. Furthermore, our analysis helped understand which service bundles should not be offered to specific groups of customers, because they do not satisfy the requirements of these customers well enough, or because other bundles can satisfy the same requirements better.

An important advantage of ontologies is that they help *reason* with domain knowledge. Our ontological approach, summarized in Figure 1, enabled reasonings as the following. The customer *want* for ‘indoor comfort’ is reduced to three wants, including ‘temperature regulation’. We found three service bundles that satisfy this want. All of them include *energy supply* plus extra services, supplied by different suppliers. In other words, these service bundles compete with each other. An electricity supplier can then decide whether to offer all of these bundles or just a subset thereof. The choice of a bundle to offer implies also a choice of a business partner to work with, since the extra services are offered by other suppliers. The same want is further specified by several demands. Reasoning on the demand level, we see that whereas the competing bundles provide

a solution to the same want, they target different demands, that encapsulate differing quality levels. A supplier can then decide whether it wants to provide all quality levels (and thus sell all generated bundles), or address only a specific target group (high-end, low-end).

## 7 Tool Support

The service offering perspective of our ontology was implemented in a CASE tool (a prototype is available at [www.cs.vu.nl/~ziv/tool](http://www.cs.vu.nl/~ziv/tool)). The tool presents an easy-to-use GUI, with which business analysts and domain experts can model services from a supplier perspective. Subsequently, the tool is capable of transforming the visually-modeled services into a computer-interpretable RDF representation, based on our service ontology, and to generate also an RDF representation of bundling requirements (i.e. in terms of resources: service inputs and service outcomes). We then use a configuration tool (based on a configuration ontology [4]) to generate service bundles based on the service descriptions and on these requirements. The tools communicate by exchanging RDF files: the service modeling tool provides service descriptions and bundling (configuration) requirements. The configuration tool provides solutions: service bundles that meet the requirements. Using our service ontology as its fundamentals, our software is capable of configuring bundles of services, when requirements are specified in terms of resources.

In the present paper we have shown how we derive such requirements: by adding an earlier step in which we formalize customer demands, and map them into available resources. We are currently involved in a project where the emphasis is put on this earlier step, for which software support will be implemented as well. Two main issues should be solved for effective software support: (1) understanding the nature of contextual information that influences production rules, and (2) conflict resolution concerning weak production rules (the *is a* relations). These issues are still ongoing work. Conflict resolution may be performed in advance, and not in real-time, because it is often desired to know in advance which bundles may be generated; in a business environment it may turn not to be desirable to enable customers to generate **any** valid service bundle, because some bundles may be valid, but yet not financially interesting for service suppliers. Consequently, a needs-driven analysis can be performed as part of a business analysis, thereby identifying and solving conflicts in advance, and then the space of possible solutions (service bundles) can be limited to those bundles that were examined and found feasible.

## 8 Conclusions and Future Work

Only a few years ago, when the 'dotcom' wave was still rolling, it was almost impossible to find an electric utility in Norway, which did not offer so-called *self-service* via its website. One could observe a great variety of products and ser-

vices, which were offered together with electricity retail contracts. Despite costly marketing campaigns, these offerings were mostly not appreciated by customers and failed. Experience shows that the bundling of services without sound logical fundamentals of the bundles-configuration process (as applied in [7]) and without reasoning about customer needs and demands may cause severe financial losses [19, 14]. The need for such fundamentals, combined with an online process, was the driving force behind our case study.

In Sections 6.3 and 6.4 we give examples of results we obtained in this case study from modeling customer demands and available services with an ontology. Domain experts declared to have gained insight into their domain by the use of an ontology to model domain knowledge. We investigated and modeled service bundles for two market segments of energy consumers: households and industrial customers. Our analysis resulted in identifying sets of services to be offered to these market segments (or: customers), to meet specific needs and demands. In other words, whereas multiple services could be offered to a customer, reasoning about his needs is required in order to satisfy the client. If this knowledge is conceptualized and formalized, software can perform this reasoning instead of humans. This conceptualization and formalization takes place in what we refer to as

Knowledge and expertise from business science, information science and computer science have been intertwined in our research to solve the problem at hand. We split the process into a customer perspective, a supplier perspective and a transformation process between the two. By expressing both perspectives using a formal ontology, also expressible in a machine-interpretable language (RDFS), we ensure that domain knowledge is formalized in such a way that it can be checked for consistency and used for reasoning by software.

Business science literature concerning customer needs acknowledges the existence of (need) hierarchies. However, it lacks a few elements, necessary for making business knowledge machine-interpretable: (1) a definition of hierarchical decompositions (e.g. AND/OR/XOR structures) of customer needs; (2) a well-defined description of services; (3) a definition of possible relations (links) between needs and solutions; and (4) an understanding of how demands (functional requirements) differ from desired service quality (non-functional requirements). As we have shown in this article, we use existing requirements engineering practices to add the necessary formalism to business concepts: we use goal hierarchies, goal links and production rules to relate features (needs, demands) to solutions (services, described by resources). By embedding these constructs and business concepts in a service ontology, expressible in a machine-interpretable language, we create a framework with which a reasoning can be performed: first customer demands trigger the selection of resources (benefits), and then a configuration process creates bundles of services that provide these customer benefits.

Our work aims at facilitating automated reasoning processes through conceptualizing and formalizing domain knowledge. Automating reasoning processes is a necessity in order to facilitate complex e-service scenarios. In this paper we show how research from various disciplines can be combined to achieve this goal.



We showed how our approach helped domain experts analyze a complex case study.

So far our research has dealt with demands. We modeled also a set of service quality criteria, but we have not explored how their analysis should differ from that of demands. As we have already seen positive results from the use of the modeling framework [24] in value-oriented modeling [21], adopting it for our current research as well seems promising.

Another future research direction concerning service quality is incorporating the SERVQUAL model [25] (which is broadly-used in business science) in the service ontology to describe service quality from a customer perspective using SERVQUAL's generic dimensions that customers use for evaluating service quality.

We have demonstrated how the mapping between the customer perspective and the supplier perspective can be performed by production rules, as modeled with FS-graphs. Future work should further investigate the nature of these production rules. We have noticed so far that some of them relate a demand – specified by (e.g. quality) properties – to a resource itself – disregarding its (quality) properties – whereas others relate a demand to a combination of a resource with (quality) properties.

## References

1. *eCI@ss website*. 2005. <http://www.eclass.de>.
2. *UNSPSC website*. 2005. <http://www.unspsc.org>.
3. Hans Akkermans, Ziv Baida, Jaap Gordijn, Nieves Peña, Ander Altuna, and Iñaki Laresgoiti. Value webs: Using ontologies to bundle real-world services. *IEEE Intelligent Systems - Semantic Web Services*, 19(4):57–66, 2004.
4. Ander Altuna, Alvaro Cabrerizo, Iñaki Laresgoiti, Nieves Peña, and Daniel Sastre. Co-operative and distributed configuration. In *Net.ObjectDays (NODE) 2004*, pages 69–80, Erfurt, Germany, 2004.
5. Ziv Baida, Jaap Gordijn, Andrei Z. Morch, Hanne Sæle, and Hans Akkermans. Ontology-based analysis of e-service bundles for networked enterprises. In *Proceedings of The 17th Bled eCommerce Conference (Bled 2004)*, Bled, Slovenia, 2004.
6. Ziv Baida, Jaap Gordijn, Borys Omelayenko, and Hans Akkermans. A shared service terminology for online service provisioning. In *Proceedings of the Sixth International Conference on Electronic Commerce (ICEC04)*, Delft, NL, 2004.
7. Ziv Baida, Jaap Gordijn, Hanne Sæle, Andrei Z. Morch, and Hans Akkermans. Energy services: A case study in real-world service configuration. In *Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE 2004)*, pages 36–50, Riga, Latvia, 2004. Springer-Verlag.
8. L.L. Berry and A. Parasuraman. *Marketing Services: Competing through Quality*. The Free Press, New York, NY, 1991.
9. E. Bigné, C. Martínez, and Maria José Miquel. The influence of motivation, experience and satisfaction on the quality of service of travel agencies. In Paul Kunst and Jos Lemmink, editors, *Managing Service Quality (Volume III)*, pages 53–70, London, UK, 1997. Paul Chapman Publishing Ltd.

10. Pim Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, Universiteit Twente, Enschede, NL, 1997.
11. Hans de Bruin and Hans van Vliet. Top-down composition of software architectures. In Per Runeson, editor, *Proceedings of 9th International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS2002)*, pages 1–10, Lund, Sweden, April 2002. IEEE Computer Society.
12. Hans de Bruin, Hans van Vliet, and Ziv Baida. Documenting and analyzing a context-sensitive design space. In J. Bosch, M. Gentleman, C. Hofmeister, and J. Kuusela, editors, *Software Architecture: System Design, Development and Maintenance; Proceedings of the 3rd Working IFIP/IEEE Conference on Software Architecture (WICSA-02)*, pages 127–141, Montreal, Canada, August 2002. Kluwer Academic Publishers.
13. Paolo Donzelli. A goal-driven and agent-based requirements engineering framework. *Requirements Engineering*, 9(1):16–39, 2004.
14. Anne Flæte and Gregers Ottesen. Telefiasko for viken. *Dagens Næringsliv* (Norwegian newspaper), 13/14 October 2001.
15. Ariel Fuxman, Lin Liu, Marco Pistore, Marco Roveri, and John Mylopoulos. Specifying and analyzing early requirements: Some experimental results. In *Proceedings of the 11th IEEE International Requirements Engineering Conference (RE'03)*, pages 105–114, Monterey Bay, California, 2003. IEEE Computer Society.
16. C. Grönroos. *Service Management and Marketing: A Customer Relationship Management Approach, 2nd edition*. John Wiley & Sons, Chichester, UK, 2000.
17. H. Kasper, P. van Helsdingen, and W. de Vries jr. *Service Marketing Management: An International Perspective*. John Wiley & Sons, Chichester, UK, 1999.
18. P. Kotler. *Marketing Management: Analysis, Planning, Implementation and Control, 6th edition*. Prentice Hall, Englewood Cliffs, NJ, 1988.
19. K. F. Marthinussen. KANKAN som kunne... (presentation). In ITEnergi 2002 Conference, 2002. Available at [www.itenergi.com/kari\\_martinussen.ppt](http://www.itenergi.com/kari_martinussen.ppt), last visited March 2005.
20. Roland T. Rust and P.K. Kannan. E-service: a new paradigm for business in the electronic environment. *Communications of the ACM*, 46(6):36–42, 2003.
21. Bas van der Raadt. Business-oriented exploration of web service ideas: Combining goal-oriented and value-based approaches. Master's thesis, Vrije Universiteit, Amsterdam, NL, 2004.
22. Axel van Lamsweerde. Requirements engineering in the year 00: a research perspective. In *Proceedings of the 22nd international conference on Software engineering*, pages 5–19, Limerick, Ireland, 2000. ACM Press.
23. Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour, invited minitutorial. In *Proceedings of RE'01 - International Joint Conference on Requirements Engineering*, pages 249–263, Toronto, Canada, 2001. IEEE.
24. Eric S. K. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Toronto, Canada, 1995. Also appears as Technical Report DKBSTR-94-6.
25. V.A. Zeithaml, A. Parasuraman, and L.L. Berry. *Delivering Quality Service: Balancing Customer Perceptions and Expectations*. The Free Press, New York, NY, 2001.

# Developing Adapters for Web Services Integration

Boualem Benatallah<sup>1</sup>, Fabio Casati<sup>2</sup>, Daniela Grigori<sup>3</sup>,  
Hamid R. Motahari Nezhad<sup>1,4</sup>, and Farouk Toumani<sup>5</sup>

<sup>1</sup> SCSE, University of New South Wales, Sydney NSW 2052, Australia  
{boualem, hamidm}@cse.unsw.edu.au

<sup>2</sup> HP Labs, Palo Alto, CA, 94304 USA  
fabio.casati@hp.com

<sup>3</sup> PriSM, Université de Versailles, 45 avenue des Etats-Unis,  
78035 Versailles Cedex, France,  
daniela.grigori@prism.uvsq.fr

<sup>4</sup> NICTA, Australian Technology Park, Bay 15 Locomotive Workshop,  
Sydney NSW 1430, Australia,

<sup>5</sup> LIMOS, ISIMA, Campus des Cezeaux, BP 125, 63173 Aubière Cedex, France  
ftoumani@isima.fr

**Abstract.** The push toward business process automation has generated the need for integrating different enterprise applications involved in such processes. The typical approach to integration and to process automation is based on the use of adapters and message brokers. The need for adapters in Web services mainly comes from two sources: one is the heterogeneity at the higher levels of the interoperability stack, and the other is the high number of clients, each of which can support different interfaces and protocols, thereby generating the need for providing multiple interfaces to the same service. In this paper, we characterize the problem of adaptation of web services by identifying and classifying different kinds of adaptation requirements. Then, we focus on business protocol adapters, and we classify the different ways in which two protocols may differ. Next, we propose a methodology for developing adapters in Web services, based on the use of mismatch patterns and service composition technologies.

## 1 Introduction

The push toward business process automation, motivated by opportunities in terms of cost savings, higher quality and more reliable executions, has generated the need for integrating the different enterprise applications involved in such processes. Application integration has been one of the main drivers in the software market during the late nineties and into the new millennium. The typical approach to integration and to process automation is based on the use of *adapters* and of *message brokers* [YeSt97, CFPT03]. Adapters wrap the various applications (which are in general heterogeneous, e.g., have different interfaces, speak different protocols, and support different data formats) so that they can appear as homogeneous and therefore easier to be integrated. Message brokers, and message-oriented middleware in general, provide an efficient and reliable way to transport messages (typically corresponding to operation invocations or

their replies) among the adapters, which in turn interact with the wrapped application. While very effective and relatively successful, this approach presents several limitations. In particular, process integration efforts require a high number of different adapters, as the level of heterogeneity in IT infrastructures is typically very high. Furthermore, whenever new versions of the wrapped applications are deployed, adapters need to be modified to account for the differences in protocols and formats supported by these new versions. This is also why enterprise application integration (EAI) platforms are very expensive.

Web services were born as a solution to (or at least as a simplification of) the integration problem [ACKM04]. The main benefit they bring is that of standardization, in terms of data format (XML), interface definition language (WSDL), transport mechanism (SOAP) and many other interoperability aspects. Standardization reduces heterogeneity and makes it therefore easier to develop business logic that integrates different (Web service-based) applications. Web services also represent the most promising technologies for the realization of service-oriented architectures (SOAs), not only within but also outside companies' boundaries, as they are designed to enable loosely-coupled, distributed interaction [BeCT04].

While standardization makes interoperability easier, it does not remove the need for adapters. In fact, although the lower levels of the interaction stacks are standardized (as discussed later), different Web services may still support different interfaces and protocols. In addition, the novel opportunities enabled by Web services have an implication in terms of adaptation needs. In fact, having loosely-coupled and B2B interactions imply that services are not designed having interoperability with a particular client in mind (as it was often the case with CORBA-style integration) [CFPT03]. They are designed to be open and possibly without knowledge, at development time, about the type and number of clients that will access them, which can be very large. The possible interactions that a Web service can support are specified at design time, using what is called a *business protocol* or *conversation protocol* [BeCT04]. A business protocol specifies message exchange sequences that are supported by the service, for example expressed in terms of constraints on the order in which service operations should be invoked. This is important, as it rarely happens that service operations can be invoked at will independently from one another. Hence, adaptation should not be limited to handling heterogeneity at the operation level, but also at the business protocol level.

This paper presents a framework for developing Web service adapters. First, we characterize the problem of adaptation by identifying and classifying different kinds of adaptation needs (Section 2). Then, we focus on interface and business protocol adapters and we classify the different ways in which two interfaces and protocols may differ (Section 3). These differences are captured using *mismatch patterns*. Patterns help users in analyzing differences and in resolving them. In fact, among other information, patterns include a template of business logic that can be used to semi-automatically develop adapters to handle the mismatch captured by each pattern. We provide a number of built-in patterns corresponding to the possible mismatch we have identified at the interface and protocol levels. Finally, we discuss related work, conclusions and future directions in Sections 4 and 5.

## 2 Toward a Methodology for Web Service Adapters

This section presents an overview of the proposed approach to semi-automated development of service adapters. We first characterize the interoperability problem in general, then we define the focus of our work, and finally we describe at a high level the approach we adopt.

### 2.1 Interoperability at Business-Level Interfaces and Protocols

Interoperability among Web services, just like interoperability in any distributed system, requires that services use the same (or compatible) protocols, data formats, and semantics. In our work, we focus on interoperability issues at business-level interfaces and protocols. To interact, services must have compatible:

- Interfaces (i.e., the set of operations supported by services),
- Business protocols (i.e., the allowed message exchange sequences). These can be expressed for example using BPEL abstract processes, WSCI, or other protocol languages (see, e.g., [BeCT04]).

More precisely, we classify the need for adaptation in Web services in two basic categories: adaptation for compatibility and adaptation for replaceability. The first category refers to wrapping a Web service *S* so that it can interact with another service *C*. For example, consider a service *S*, offered by provider *SP*, allowing companies to order office supplies. If *SP* wants to be able to do business with certain retailers (say, *Wal-Mart* or *Target*), then it needs to adapt its service *S* so that it can interoperate with these retailers. In general, many adapters can be defined depending on the number of different client protocols that *SP* must interact with. Hence, in this case, adaptation is performed by considering the client's protocol. Note that adaptation may be required for one or more of the interoperability layers identified above, since for two services to interact, compatibility must be achieved at all layers.

Adaptation for replaceability refers to modifying a Web service so that it becomes *compliant* with (i.e., can be used to replace) another service. This is important especially in those business environments where the interaction, even at the interface and business protocol level, has been standardized either *de jure* or *de facto* (e.g. due to the presence of a dominant player in the market). For example, the RosettaNet consortium standardizes the external behaviour of services in the IT supply chain space. In these cases, service providers may have to adapt their services so that they can follow the guidelines prescribed by the standards.

Adaptation for replaceability is also needed when a new version of a service is developed, possibly with a different external behaviour, but we want to preserve backward compatibility (that is, an adapter should be provided so that the service is also offered in a version that behaves like the old one). Replaceability may be *partial* or *total* [BeCT04a]. Total replaceability occurs when a service *SR* behaves externally like another service *S*. This means that any service that interacts *correctly* (i.e., without generating runtime faults) with *S* will also be able to interact correctly with *SR* (note that the opposite is not necessarily true). Partial replaceability occurs when a service *SR* can behave like *S* only in certain interactions (that is, *SR* behaves like *S* in some but not all conversations). For example, an ordering service *SR* may need or be

able to replace ordering service S only for orders of certain products but not other, or may able to process all orders but does not allow cancellations, while S does. We refer the reader to [BeCT04a] for a detailed definition of compatibility and replaceability among services, as well as other important relations among different elements of service descriptions.

This paper proposes a technique for developing adapters to achieve total replaceability. As mentioned above, this is a very important and relevant problem. The related issues of partial replaceability and compatibility can be handled in an analogous manner. We also decided to initially focus on developing adapters to resolve differences at the interface and business protocol level. Replaceability at the lower layers has been either addressed by standardization (e.g., messaging) or has been the subject of excellent research work by other groups [RyWo], and hence research on protocol replaceability constitutes the next level up the interoperability stack in supporting interaction among services. Incidentally, although we discuss the problem of protocol replaceability in the context of business protocols, analogous techniques can be used for other service aspects characterized by protocols (e.g., trust negotiation protocols or basic coordination).

### 2.2 Developing Service Adapters Using Mismatch Patterns

The intended benefit of this work is to help programmers develop adapters through a methodology and semi-automated code development, starting from the protocol definitions. The adapters have the goal of making a service SR, characterized by protocol PR, "look like" (interact as) another service S that has protocol P, so that SR can then interact with any client that can interact with S (see Figure 1).

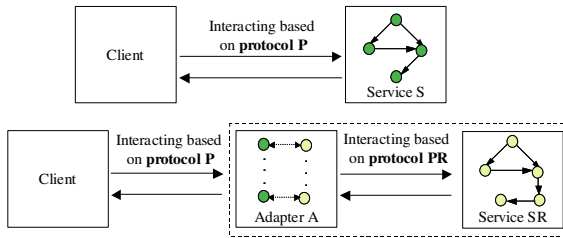


Fig. 1. Adapters allows achieving protocol replaceability

Hence, the adapter A for SR is a Web service that, to clients, behaves like S from an interaction perspective. In particular, if S supports protocol P, then adapter A also supports protocol P when interacting with clients. Adapter A will implement protocol P by invoking methods of SR. From the perspective of SR, A looks like a service whose protocol is *compatible* with PR (Figure 1).

The approach proposed in this paper to adapter development is based on mismatch patterns, which are design patterns that can be used to capture the possible differences among services (and specifically among interfaces and protocols). We have analyzed interfaces and protocols to identify common differences and for those we have specified the corresponding mismatch patterns. Indeed we believe that the identification of

the various kinds of differences among interfaces and, most of all, protocols, is a contribution in itself. Developers can, however, add to the set of patterns if there are specific mismatches that they would like to handle differently or if there are mismatches that are not captured in the built-in set.

Besides capturing differences, patterns can be used both as guidelines for designer in developing adapters and as input to a tool that automatically generates the adapter code. In fact, mismatch patterns contain both formal and informal descriptions of the type of adapter (called *adapter template*) used to resolve that type of mismatch. The table below summarizes the structure of an adapter pattern. In the following of this section we discuss and exemplify in more detail the part related to adapter templates and their instantiation.

Name	Name of the pattern
Mismatch Type	A description of the type of difference captured by the pattern
Template parameters	Information that needs to be provided by the user when instantiating an adapter template to derive the adapter code
Adapter template	Code or pseudo-code that describes the implementation of an adapter that can resolve the difference captured by the pattern
Sample usage	The sample usage section contains information that guides the developer in customizing (or manually generating) the adapter, by providing examples on how to instantiate the template

To describe the approach to adapter template specification and adapter generation and to motivate our choices, we begin by discussing what is expected of an adapter and how they can be modelled and implemented. As mentioned above, the job of an adapter consists in mapping interactions with protocol P into interactions with protocol PR. This requires performing activities such as receiving messages, storing messages, transforming message data, and invoking service operations. These tasks can be very well modelled by process-centric service composition languages such as BPEL (<http://www-128.ibm.com/developerworks/library/ws-bpel/>). Hence, our aim is to leverage patterns to manually or automatically generate process skeletons that map interactions according to protocol P into interactions according to protocol PR. Analogous solutions can be identified if third-generation programming languages like Java or C# or if other process languages instead of BPEL are used. In any case, the generated specifications can be then enacted by the corresponding execution engine (e.g., a BPEL engine, or a Java virtual machine). We chose a process-based notation because it is well-suited to model business logic and because it is easy to derive the protocol specifications of a service when its implementation is specified as a business process [BBCT04] (although, as we will see, other aspects such message transformations need to be modelled). A high-level process-based notation is also appropriate to compose complex adapters from primitive adapter templates, possibly leveraging one of the many process management tools available on the market. In addition, this notation can be mapped to others (e.g., state machines or state-charts), which are endowed with formal semantics. Using a high level notation allows, e.g., using formal analysis techniques to verify the correctness of adaptors.

Given that we aim at generating and customizing a process definition, it was natural to select a process language for defining the adapter templates as well, as using a

similar modelling framework simplifies adapter generation, especially when it is performed manually. Indeed, we borrow BPEL notation, concepts, and terminology for this purpose, endowed with additional annotations to specify *adaptation abstractions*. In particular, the additional annotations may include XQuery ([www.w3.org/TR/xquery/](http://www.w3.org/TR/xquery/)) functions to specify message transformations that are commonly needed in adapters, directives to help developers understand how to instantiate certain elements of the adapter template.

**Example.** As an example, consider the MapPoint ([www.microsoft.com/mappoint/](http://www.microsoft.com/mappoint/)) and Arcweb ([www.esri.com/software/arcwebservices/](http://www.esri.com/software/arcwebservices/)) route Web services, which offer similar functionalities for finding driving routes between two points using different WSDL interfaces (operations `CalculateRoute` and `findRoute`, respectively). Suppose that Arcweb corresponds to service SR and MapPoint to service S according to the architecture presented in Figure 1<sup>1</sup>. The names, number, and types of the input/output parameters of the operations `CalculateRoute` and `findRoute` differ. The operation `CalculateRoute` requires one input parameter called `Specification` whose type is `SegmentSpecification`. The operation `findRoute` requires two parameters: `routeStops` and `routeFinderOptions` whose types are `RouteStops` and `RouteFinderOptions`, respectively. The values of both parameters `routeStops` and `routeFinderOptions` can be computed from the value of the parameter `Specification`.

This type of difference is handled by a mismatch pattern called *SMP (Signatures Mismatch Pattern)*. This pattern concerns differences that occur when two services S and SR have operations that have the same functionality but differ in operation name, number, order or type of input/output parameters. In general, adapters that resolve this kind of differences need to perform the actions described below, which therefore constitute our adapter template for this pattern (written here in pseudo-code for ease of presentation):

Template Parameters	<b>Signatures of operations O of service S and OR of service SR, XQuery functions for message transformations</b>
Adapter Template	<p><i>Receive</i> the input message OI of operation O from client (<i>BPEL receive activity</i>)</p> <p><i>Transform</i> OI into a format that is compliant with the type of input message ORI of operation OP, using XQuery transformation functions (one or more <i>BPEL assign activity</i>, depending on the parameters to be transformed)</p> <p><i>Invoke</i> operation OR (<i>BPEL invoke activity</i>)</p> <p><i>Transform</i> output message ORO of operation OR into a format that is compliant with the type of the output message OO of operation O, using XQuery transformation functions (one or more <i>BPEL assign activity</i>, depending on the parameters to be transformed)</p> <p><i>Send</i> reply of operation O to client (<i>BPEL reply activity</i>)</p>

Note that the template is parametric: to instantiate it and generate an executable BPEL process, the user needs to provide several parameters. In this case, the parameters are the signatures of the operations that have a mismatch and the XQuery trans-

---

<sup>1</sup> The idea for using the arcweb vs MapPoint example is taken from [PoFo04].



formation functions. The parameters that the user needs to specify are part of the *template parameters* field of the pattern. This information is used to (manually or automatically) generate a process skeleton from the template. The developer may then want or need to further customize the resulting process skeleton to add some custom business logic, or can just directly use the generated process skeleton to deploy the adapter. For built-in patterns, we have automated code that actually generates adapters given the pattern name and template parameters. Note that complex adapters, i.e., those resolving several mismatch types, can be constructed by composing primitive templates.

Figure 2 shows an adapter that resolves the signature mismatch among operation CalculateRoute of S and operation findRoute of SR according to the adapter template of SMP pattern.

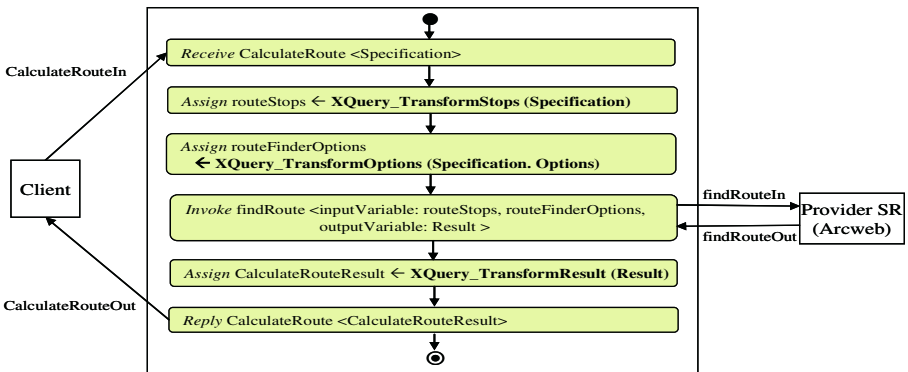


Fig. 2. Sample usage of SMP

In the process skeleton, the parts in bold indicate the parameters that are provided by the adapter developer. The symbol "<>" is used to denote parameters of operations. We also identify the input and output messages of operations by adding "In" and "Out" to the end of operation name in the examples.

In Figure 2, the adapter first receives a message that contains the value of the input parameter Specification of operation CaculateRoute (hence behaving like S). Then it computes the values of routeStops and routeFinderOptions (i.e, input parameters of the operation findRoute) from the value of the parameter Specification, via XQuery transformation functions. These functions are specified by the developer, possibly by using one of the many XQuery tools being developed by major software vendors. After performing message transformations, the adapter invokes the operation findRoute of SR (Arcweb), and performs symmetric actions on the reply. In this example, the configuration of the template activities consists of specifying XQuery functions, namely XQuery\_TransformStops, XQuery\_TransformOptions and XQuery\_TransformResults.

We conclude the section by pointing out aspects that are outside the scope of this work. The work in this paper does not address the problem of automatically identifying differences between two actual protocols. For example, a type of difference occurs when protocol P requires two messages,  $m_a$  and  $m_b$  to be in sequence, while PR allows

them to be in any order. Referring to this example, we do not present here a mechanism for finding out that the difference between P and PR consists in the different ordering constraints on  $m_a$  and  $m_b$ . The problem of identifying the actual differences constitutes a separate research thread in itself and is extremely complex. This is, however, an orthogonal issue. In this paper, we assume that an analyst will identify the differences and the corresponding pattern. For example, an analyst (or, in the future, a tool) will look at P and PR and identify that there is a difference of type ordering constraint involving  $m_a$  and  $m_b$ . From there, we derive the corresponding mismatch pattern that resolves the difference so that the adapter can appear as supporting protocol P and is implemented by invoking operations of PR.

### 3 Characterizing and Resolving Differences Between Business Protocols

This section describes our approach for developing Web service adapters at the interface and business protocol levels. For each level, we present a taxonomy of possible mismatches and propose a solution to tackle each kind of mismatch. The rationale behind the proposed taxonomy is to characterize differences based on how we can approach/solve them. For each pattern we also provide an example (which corresponds to a sample usage for that pattern).

#### 3.1 Differences at the Operation Level

In addition to SMP pattern that we described in section 2.3 for resolving operation signatures mismatch, in this section we describe a mismatch pattern called *PCP (Parameter Constraints Pattern)* that handles parameter constraints mismatch as described below. This type of mismatch occurs when the operation O of S imposes constraints on input parameters, which are less restrictive than those of OR input parameters in SR (e.g., differences in value ranges).

Template Parameters	<b>Signatures of operations O of service S and OR of service SR, XQuery functions for checking parameter constraints</b>
Adapter Template	<p><i>Receive the input message OI of operation O from client (BPEL receive activity)</i></p> <p>If OI verifies OR constraints (<i>BPEL switch activity</i>):</p> <p>Then <i>Invoke operation OR (BPEL invoke activity)</i></p> <p><i>Send reply of operation O to client (BPEL reply activity)</i></p> <p>Else <i>Raise a constraint-violation exception and terminate conversation (BPEL reply activity)</i></p>

In this adapter template, input messages of operation O are first checked to verify if they are compliant with OR constraints. For instance, suppose that element Preference (a sub-element of the parameter Specification of operation CalculateRoute) accepts "quickest", "shortest" and "Least Toll" as possible values. But, element RouteType (an element of parameter routeFinderOptions of operation findRoute) accepts "quickest" and "shortest" as possible values. In this case, there is no possible value of RouteType that corresponds to the value "Least Toll" of Pref-

erence. If the value of `RouteType` is in {"quickest", "shortest"}, the adapter will forward the invocation message to Arcweb. Otherwise, the adapter will raise a constraint violation exception. Figure 3 shows an adapter resolving this constraint mismatch. We observe again that in the case of built-in patterns we have pattern-specific code that generates the adapter given the user-defined parameters. However, the same can be done manually by the developer by looking at the adapter template field of the pattern and at the sample usage. Note that constraint-checking conditions is expressed using XQuery queries. For example, the condition `Verify_Specifica tion_Constraints` checks if `Specification` verifies the constraints of `RouteType`.

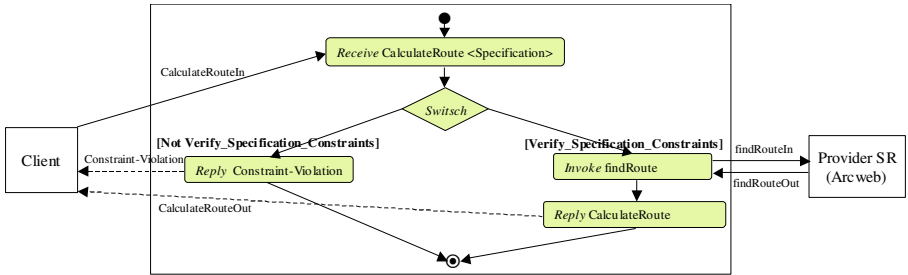


Fig. 3. Sample usage of PCP

### 3.2 Differences at the Protocol Level

We now consider the problem of developing adapters to resolve service mismatches that occur at the protocol level. We build our approach using the extend business protocol model presented in [BeCT04]. That protocol model allows a richer description of the external behavior of a service by providing specific abstractions that enable, for example, to model temporal and transactional properties of service conversations.

In this section we use a supply chain example to illustrate adapter templates. For instance, protocol P may expect to exchange messages in the following order: clients can invoke `login`, then `getCatalogue` to receive the catalogue of products including shipping options and preferences (e.g., delivery dates), followed by `submitOrder`, `sendShippingPreferences`, `issueInvoice`, and `makePayment` operations. In contrast, protocol PR allows the following sequence of operations: `login`, `getCatalogue`, `submitOrder`, `issueInvoice`, `makePayment` and `sendShippingPreferences`. This is possible, e.g., because provider SR does not charge differently according to the shipping preferences. Clients are allowed to specify their shipping preferences at a final step. Note that for the sake of clarity, we omitted the acknowledgements from the message sequences.

#### Message Ordering Mismatch

This type of difference is concerned with the order in which protocols expect to receive certain message. Mismatch occurs when protocols P and PR support the same message but in different orders. This type of difference is handled by a mismatch pattern called *OCP (Ordering Constraint Pattern)* described below:

Template Parameters	<b>Protocols P and PR, message m to be re-ordered</b>
Adapter Template	<i>Perform activities as prescribed by P for parts that do need adaptation (BPEL receive, invoke, reply activities)</i> <i>Receive message m according to protocol P (BPEL receive activity)</i> <i>Store m in the adapter (BPEL assign activity)</i> <i>Send m to SR when it is expected (BPEL invoke activity)</i>

Figure 4 shows an adapter that resolves the ordering constraints for the message `sendShippingPreferencesIn`. From the input parameters of the template, it is possible to determine the message ordering constraints of protocols P and PR. In this case, the adapter can temporarily store the parameter of operation `sendShippingPreferences` of protocol P and forward the operation to service SR according to the messages choreography of protocol PR. Note that the adaptation will be more complex if protocol P issues a reply or acknowledgement for the `sendShippingPreferencesIn` message and the client expect receiving such message. We will discuss such a scenario in the following when we discuss the need for generating missing messages.

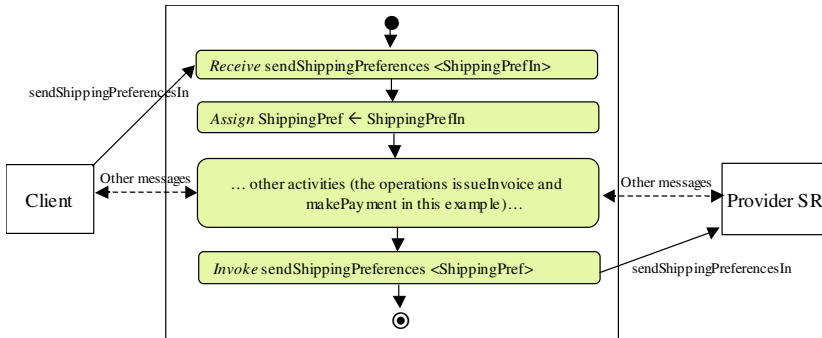


Fig. 4. Sample usage of OCP

**Extra Message Mismatch**

This type of differences occurs in situations where protocol PR issues an extra message that protocol P does not issue. This type of difference is handled by a mismatch pattern called *EDP (Extra Message Pattern)*. The adapter template of EDP allows intercepting and discarding the extra message in order to make PR look like P. It should be noted that such an adaptation makes sense only if the extra message of PR does not affect the semantics of the target protocol (i.e., does not change the functionality of PR).

Template Parameters	<b>Protocols P of S and PR of SR, message m of PR to be discarded</b>
Adapter Template	<i>Perform activities as prescribed by P for parts that do need adaptation (BPEL receive, invoke, reply activities)</i> <i>Discard m when received (BPEL receive activity)</i>

In the supply chain scenario, assume that protocol PR sends an acknowledgement after receiving message `issueInvoiceIn` but protocol P does not. Figure 5 shows an adapter that when receives the message `InvoiceAck`, it discards it (does not perform any action).

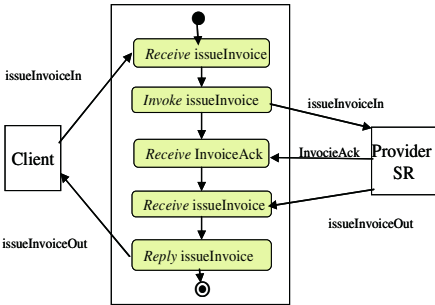


Fig. 5. Sample usage of EMP

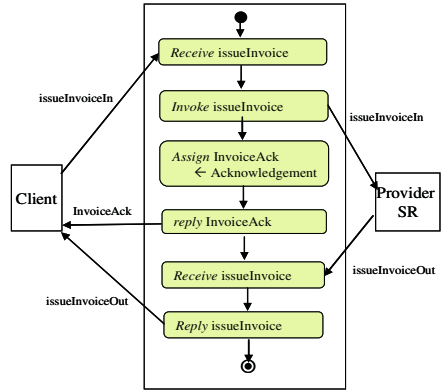


Fig. 6. Sample usage of MMP

**Missing Message Mismatch**

This type of differences occurs when protocol P issues an extra message that protocol PR does not issue. It should be noted that this extra message does not affect the semantics of the protocol PR. This type of difference is handled by a mismatch pattern called *MMP (Missing Message Pattern)*. The adapter template of MMP generates a new message to make PR look like P.

Template Parameters	Protocols P of S and PR of SR, message m of P to be generated
Adapter Template	Perform activities as prescribed by P for parts that do need adaptation (BPEL receive, invoke, reply activities) Generate m when expected by P (BPEL assign activity) Reply m according to P (BPEL reply activity)

Consider the opposite case of the previous example, where protocol P issues an acknowledgement when receiving a request for invoice (i.e., the message `issueInvoiceIn`), while protocol PR does not. Figure 6 shows an adapter that generates the message `InvoiceAck` and sends it to the client after receiving the message `issueInvoiceIn` and invoking the operation `issueInvoice` of SR.

**Message Split Mismatch**

This type of differences occurs when the protocol P requires a single message to achieve certain functionality, while in protocol PR the same behavior is achieved by receiving several messages. This type of difference is handled by a mismatch pattern called *OMP (One to Many messages Pattern)* described below:

Template Parameters	<b>Protocols P and PR, message m of P to be split and messages mr<sub>1</sub>, ..., mr<sub>n</sub> of PR to be extracted from m, XQuery functions for parameters extraction</b>
Adapter Template	<i>Perform activities as prescribed by P for parts that do need adaptation (BPEL receive, invoke, reply activities)</i> <i>Generate mr<sub>1</sub>, ..., mr<sub>n</sub> from m when m is received, send mr<sub>1</sub>, ..., mr<sub>n</sub> as prescribed by PR (BPEL assign, invoke activities)</i>

Suppose that protocol P requires to receive the purchase order as well as shipping preferences in one message called submitOrderIn, while protocol PR needs two separate messages for this purpose, namely, sendShippingPreferencesIn and submitOrderIn. Figure 7 shows an adapter that resolves this mismatch. In this case, when the adapter receives submitOrderIn, it generates the parameters of the operations sendShippingPreferences and submitOrder of PR from the parameter of the operation submitOrder of P, using XQuery transformation functions, namely XQuery\_SplitShipping and XQuery\_SplitOrder. Following that messages submitOrderIn and sendShippingPreferencesIn are forwarded to SR. Note that in other cases, since messages are stored in the adapter, the adapter can forward them to SR in the order prescribed by PR.

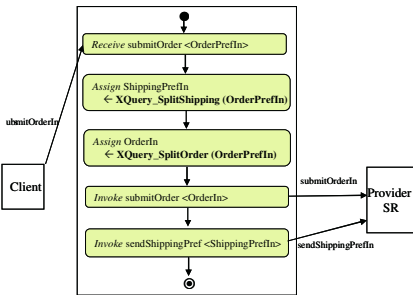


Fig. 7. Sample Usage of OMP

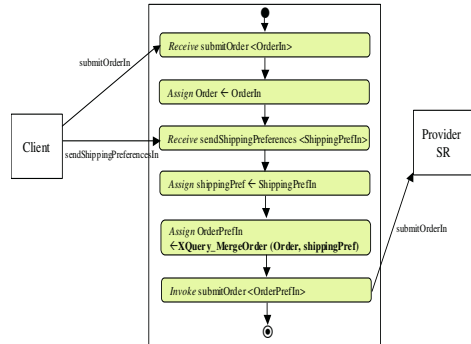


Fig. 8. Sample usage of MOP

**Message Merge Mismatch**

This type of differences occurs when protocol P needs to receive several messages for achieving certain functionality while protocol PR requires one message to achieve the same functionality. This type of difference is handled by a mismatch pattern called MOP (Many to One message Pattern) described below.

Template Parameters	<b>Protocols P and PR, messages m<sub>1</sub>, ..., m<sub>n</sub> of P to be merged into message mr of PR, XQuery function for parameter computation</b>
Adapter Template	<i>Perform activities as prescribed by P for parts that do need adaptation (BPEL receive, invoke, reply activities)</i> <i>Receive m<sub>1</sub>, ..., m<sub>n</sub> according to P (BPEL receive activities) and store them until mr is generated (BPEL assign activities)</i> <i>Generate mr by merging m<sub>1</sub>, ..., m<sub>n</sub> when mr is expected by PR (BPEL assign, invoke activities)</i>

Suppose that protocol P requires messages `submitOrderIn` and `sendShippingPreferencesIn` separately, but protocol PR needs all of this information included in the `submitOrderIn` message. Figure 8 shows an adapter that resolves this mismatch. In this template, when the adapter receives the messages `submitOrderIn` and `sendShippingPreferencesIn`, it generates the parameter of operation `submitOrder` of PR using an XQuery function, namely `XQuery_MergeOrder` that merges the parameters of the operations `submitOrder` and `sendShippingPreferences` of P. The adapter knows the order of messages of protocols P and PR from the input, so it is able to generate receive and storage activities for messages and to invoke operations of SR according to PR.

It should be noted that differences can be complex (i.e., cannot be reduced to one of the above primitive patterns). Adapters resolving several mismatch types can be constructed by composing primitive templates. We plan to extend the set of protocols management operators provided in our framework and reported in [BeCT04a] to cater for adapter templates composition.

## 4 Related Work

Although a lot of work and progress has already been done in the area of web services in the last few years, efforts have been mostly focused on service description models and languages, and on automated service discovery and composition [ACKM04].

Very recently, authors in the academia have published papers that discuss similarity and compatibility at different levels of abstractions of a service description (e.g., [BeCT04a, Bordeaux04, DHMN+04, PoFo04, WMFN04]). In terms of protocols specification and analysis, existing approaches provide models (e.g., based on pi-calculus or state machines) and mechanisms to compare specifications (e.g., protocols compatibility and replaceability checking). In particular, the work that is more related to ours, also in terms of overall line of research, is that of Lenezirini and Mecella's group. Specifically, in [Bordeaux04] a protocol analysis framework based on concepts analogous to those of replaceability and compatibility is presented. In [PoFo04], the authors proposed a framework for handling differences among service interfaces, but protocols are not discussed.

The framework we propose in this paper builds upon this previous work as well as on work we did over the past three years in the area of service protocols modeling, analysis, and management to classify differences among service protocols providing similar functionalities and bridge these differences via adapters (see [BeCT04, BeCT04a] for representative examples of our line of research).

Our aim is to provide a comprehensive framework for managing differences at various abstraction layers including interface, protocol, and policy aspects. In this paper, we focus on both interface and protocol levels. To the best of our knowledge, there is no existing work that considers the classification and management of differences between service protocols and the development of adapters to resolve them.

In the software engineering area, few approaches exist for analyzing software components mismatch. In [ZaWi97], the authors focus on analyzing differences related to data types and to pre- and post-conditions in component interfaces. In [YeSt97, CFPT03] the focus is on specifying interface mappings and using these

mappings to build component adaptors. These efforts provide mechanisms that can be leveraged for Web service protocols adaptation, but are not sufficient. In fact, service protocols require richer description models than component interfaces. This is because clients and services are typically developed by separate teams, possibly even by different companies, and service descriptions are all client developers have to understand to know how the service behaves.

## 5 Discussion and Ongoing Work

We argue that, while standardization is crucial in making service oriented computing a reality, the effective use and widespread adoption of service technologies and standards requires high-level frameworks and methodologies for supporting automated development and interoperability (e.g., mechanisms for analyzing protocol compatibility, replaceability and compliance, semi-automated generation of adaptors). We see the evolution of the work in Web services interoperability as mirroring in a way, at least conceptually, the work done in databases over the last thirty years and leading to standard models and languages, algebras, theoretical foundations, and transformation techniques. We believe that Web services protocols require the same kind of background work in terms of simple models, operators, algebras, and support for manipulation/transformation. The work we have been doing goes in this direction, and the research presented in this paper is a key part of it. Indeed. The framework presented in this paper is one of the components of a broader CASE tool, partially implemented, that manages the entire service development lifecycle.

In this paper, we focused on identifying differences among heterogeneous business protocols in Web services and on semi-automatically resolving such differences when possible via adaptors. We have identified mismatch patterns as a convenient mean to capture the differences among interface/protocols of two services and to encapsulate the solution to such differences. This solution is in the form of process fragments that can be manually or automatically instantiated to generate actual adaptors. Other components of our framework focus on modeling and specifying service abstractions: composition logic, business protocols, and trust negotiation policies [BeCT04, SBC03]. Based on these models, service lifecycle can be automated, from development through analysis, management, and enforcement [BBCT04, BeCT04a]. We believe that this work will result in a comprehensive methodology and platform that can support large-scale interoperation of Web services and facilitate service lifecycle.

As the reader will have noticed, there are several aspects that we have not discussed in this paper, some due to lack of space, others because we have not developed a solid, validated solution as yet. An issue is that of automated code generation in relation to the adapter template. In the current framework, the code for generating the adapter is pattern-specific, that is, each pattern comes with an associated function that takes actual values for the pattern parameters as input and generates the adapter code. If a user wants to develop a new pattern, then the functions must also be provided. In reality, in the current version of the framework these functions do not access the adapter templates, which can therefore be specified semi-formally (its main purpose right now is to help the developer understand the pattern and its solution, especially in case a manual adapter generation is needed). In the future we plan to define a formal



language (an extension of BPEL) that can be used to formally specify adapter templates. Then, generic (as opposed to pattern-specific) code will generate adapters by reading the adapter template. As mentioned, we use annotated BPEL for this purpose, but at the time of writing we have not finalized the language design nor the generation code. We believe this will be an improvement as pattern developers can use this BPEL-like language to specify the template and do not need to write low-level code for adapter generation.

## References

- [ACKM04] Alonso, G., Casati, F., Kuno, H., Machiraju, V.: *Web Services: Concepts, Architectures, and Applications*. Springer Verlag (2004).
- [BBCT04] Baina, K., Benatallah, B., Casati, F., Toumani, F.: *Model-Driven Web Service Development*. Procs of CAiSE'04, Riga, Latvia (2004).
- [BeCT04] Benatallah, B., Casati, F., Toumani, F.: *Web services conversation modeling: A Cornerstone for E-Business Automation*. IEEE Internet Computing, vol. 8, no. 1 (2004).
- [BeCT04a] Benatallah, B., Casati, F., Toumani, F.: *Analysis and Management of Web Services Protocols*. ER'04, Shanghai, China (2004).
- [Bordeaux04] Bordeaux, L., et al: *When are two Web Services Compatible?*. VLDB TES'04. Toronto, Canada (2004).
- [CFPT03] Canal, C., Fuentes, L., Pimentel, E., Troya, J., Vallecillo, A.: *Adding Roles to CORBA Objects*. IEEE TSE, vol. 29, no. 3 (2003).
- [DHMN+04] Dong, X., A., Halevy, Y., Madhavan, J., Nemes, E., Zhang, J.: *Similarity Search for Web Services*. VLDB'04. Toronto, Canada (2004).
- [PoFo04] Ponnekanti, S. R., Fox, A.: *Interoperability among Independently Evolving Web Services*. Middleware'04. Toronto, Canada (2004).
- [RyWo] Ryan, N. D., Wolf, A. L.: *Using Event-Based Translation to Support Dynamic Protocol Evolution*. ICSE'04. Edinburgh, Scotland, United Kingdom (2004).
- [SBC03] Skogsrud, H., Benatallah, B., Casati, F.: *Model-Driven Trust Negotiation for Web Services*. IEEE Internet Computing, vol. 7, no. 6 (2003) 45-52.
- [YeSt97] Yellin, D. M., Strom, R. E.: *Protocol specification and Component adaptors*. ACM TOPLAS, vol. 19, no. 2 (1997).
- [WMFN04] Wombacher, A., Mahleko, B., Fankhauser, P., Neuhold, E.: *Matchmaking for Business Processes based on Choreographies*. EEE'04, Taipei, Taiwan (2004).
- [ZaWi97] Zaremski, A. M., Wing, J. M.: *Specification Matching of Software Components*. ACM TOSEM, vol. 6, no. 4 (1997).

# Efficient: A Toolset for Building Trusted B2B Transactions

Amel Mammari<sup>1</sup>, Sophie Ramel<sup>2</sup>, Bertrand Grégoire<sup>2</sup>,  
Michael Schmitt<sup>2</sup>, and Nicolas Guelfi<sup>1</sup>

<sup>1</sup> Software Engineering Competence Center (SE2C), University of Luxembourg,  
6, rue Richard Coudenhove-Kalergi L-1359 Luxembourg-Kirchberg, Luxembourg  
{amel.mammari, nicolas.guelfi}@uni.lu

<sup>2</sup> Centre d'Innovation par les Technologies de l'Information (CITI),  
Centre de Recherche Public Henri Tudor,  
29, Avenue John F. Kennedy L-1855 Luxembourg-Kirchberg, Luxembourg  
{sophie.ramel, bertrand.gregoire, michael.schmitt}@tudor.lu

**Abstract.** The paper introduces an approach to the specification, the verification and the validation of B2B transactions. Based on the usage of a subset of formally defined UML diagrams complemented with business rules, we introduce two facilities offered by the supporting Efficient toolset, namely the checking of formal properties expected from the produced models as well as the animation tool allowing business experts to understand and 'play' with business transactions models before they are implemented. The overall approach is illustrated through the experiences gained in the performance of a real transactional Import/Export business case.

## 1 Introduction

Message-based Electronic Data Interchange as specified by industry standards such as EDIFACT, EDIFICE, ANSI X12 or VDA is progressively being replaced by transaction-based and extensible specifications covering the needs of the whole business process. XCBL [23], ebXML [3] and Rosettanet PIPs [14] are a few examples of new recommendations that illustrate this trend. These initiatives are defining a set of harmonized business scenarios along with the respective rules, messages and technical requirements that facilitate their implementation. Trading partners that are planning to set up an E-Commerce transaction can develop their own scenario by selecting and adapting an existing standard in accordance with their business needs. The basic building blocks of an E-Commerce scenario are the flow and the rules according to which business documents (messages) are exchanged among the actors participating in the transaction. Together with this new business transaction perspective is also adopted XML[6] and a set of associated non-proprietary technologies as the new syntax for representing the exchanged messages.

Regarding the design of the B2B transactions new recommendations like those included in the UMM method [17] proposes the use of UML[18] in order to produce a set of precise models of the transaction. However, as already experienced by many

practitioners, some UML diagrams are not enough precise to have a unique interpretation. Furthermore all the semantics of a B2B transaction cannot be captured in terms of UML diagrams that need to be complemented with additional descriptions expressed in natural language. To overcome these problems we have developed a specific CASE tool supporting the design of fully formal UML based models. Associated with these models, and taking direct profit of the absence of ambiguities in them, we have developed two additional facilities, which allow to discover errors in these models, namely

- A verification tool which, based on a proof system, allows to check the consistency and the completeness of the UML models.
- A validation tool which, based on an animator system, allows business experts to understand and ‘play’ business transactions before they are implemented.

These two facilities allow to discover errors in the UML models at the design time rather than at the implementation and test time. The direct consequence is not only a gain of resources due to an early discovery of errors already at the design time of the transaction, but also an enhancement of trust from business experts into the produced models.

The Efficient approach differs from related work ([26], [27]) in that the business requirements formulated and modeled by a group of business experts are first validated automatically to check that they contain no logic errors, and then translated into an animation that permits the business experts themselves to validate that the transaction models correspond to their business needs.

The rest of the paper is structured as follows. In Section 2, we present the overall approach supported by the Efficient CASE tool. In Section 3, we then focus on the specification layer and illustrate its purpose by means of a case study illustrating a real B2B Import/Export business case. Section 4 details the business rules language that is used in complement to the UML models. Finally, verification and validation facilities are described in Section 5. Section 6 wraps up with conclusions and future directions.

## 2 Overview of the Efficient Process

Underlying the Efficient CASE tool is a proposed methodology, which makes three layers of descriptions distinct, as illustrated in Fig. 1:

- The *business layer* provides a top-level view on the business scenario that governs the transaction. It depicts the transaction configuration and allows the trading partners to develop a common understanding of the business goals, of the roles and responsibilities that apply, of the various business activities it involves and of each actor’s contributions and benefits. In sum, this layer addresses the question of who exchanges what with whom and expects what in return. The main actors along with their respective activities are modeled by a UML use case diagram associated with scenarios capturing the transaction. Furthermore, a global UML class diagram captures the core business concepts and their relationships that are at the heart of the information exchange. We refer to this class diagram as the *business domain* underlying the transaction.

- The *specification layer* details the flow and the business documents (messages) that the trading partners exchange in the transaction. We use UML activity diagrams to model the flow of the transaction and UML class diagrams to specify the content of a business document or message. Each document regroups a subset of the elements of the business domain depending on the information needs related to the business activity the recipient of a message will need to act upon. Each message may be further annotated with a set of business rules (expressed both in an informal and formal way).
- At the *technical layer*, after the transaction has been formally verified, the business transaction is executed using a workflow engine. The infrastructure that this layer is based upon (workflow engine, interfaces, XML messages, rules checker) is automatically configured and generated on the basis of the models developed in the other two layers. The architecture of the Efficient tool set is distributed and Internet-based, allowing trading partners from different sites to participate in the execution of the transactions using a Web browser interface.

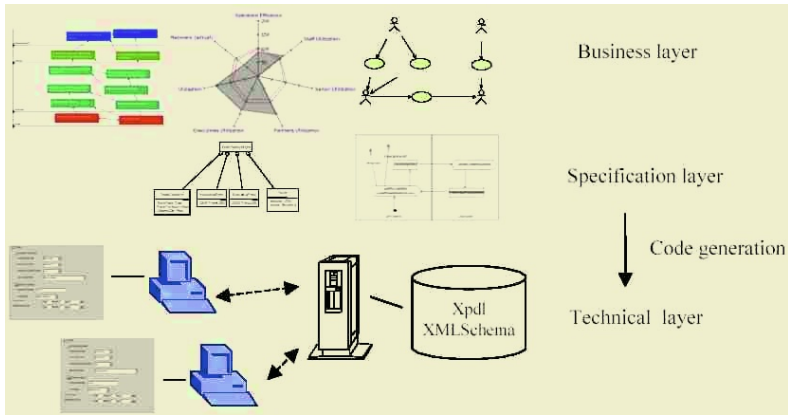


Fig. 1. The modeling layers of the Efficient tool set

More information about the business layer can be found in [16]. Details about the technical layer and more specifically about automated generation of code from UML models are available in [4]. The rest of this paper focuses on the specification layer with a specific focus on the proposed business rules.

### 3 The Specification Layer

The following sections presents the specification of the static and dynamic aspects of an e-business transaction in the form of UML diagrams, which requires a joint effort of both the IT experts who understand the requirements of the formal modeling languages and the business experts who help to formulate the requirements the transaction must meet. To create these models, the Efficient framework uses a

commercial UML CASE tool called MagicDraw [11] to which a plug-in has been developed that caters for the generation and validation activities discussed in the next sections.

In the rest of the paper we illustrate our ideas on a real business case that we have performed within the context of the EU ‘‘EA2’’ project (Euro-Asian EDI Integration), whose goal is to help the country of Vietnam with the appropriation of international EDI standards to the particular needs of their socio-economic context. The transaction we selected in this case study is based on a European initiative to standardize data exchange between customs (New Computerized Transit System). It defines the various data exchanges required in an Import/Export transaction.

### 3.1 Business Domain

To begin with, the actors implied in the global transaction must be identified, as well the different associated transactions. In the NCTS import/export transaction, the following roles have been identified:

- The Principal Trader: the party that sends the goods
- The Destination Trader: the party that receives the goods
- The office of Departure: the customs office in the country of departure
- The office of Destination: the customs office in the country of destination
- The office of Transit: the customs office in a country where the consignment is supposed to cross the external border(s)
- The office of Guarantee: the office in the country of departure that registers and maintains a guarantee for the goods listed under the export agreement.

Then the key concepts of the business sector, what we refer to as the business domain, are identified and documented in a class diagram This diagram contains classes without operations and hence consists only of classes, attributes and relations. A part of the business domain of our transaction is represented on Fig. 2.

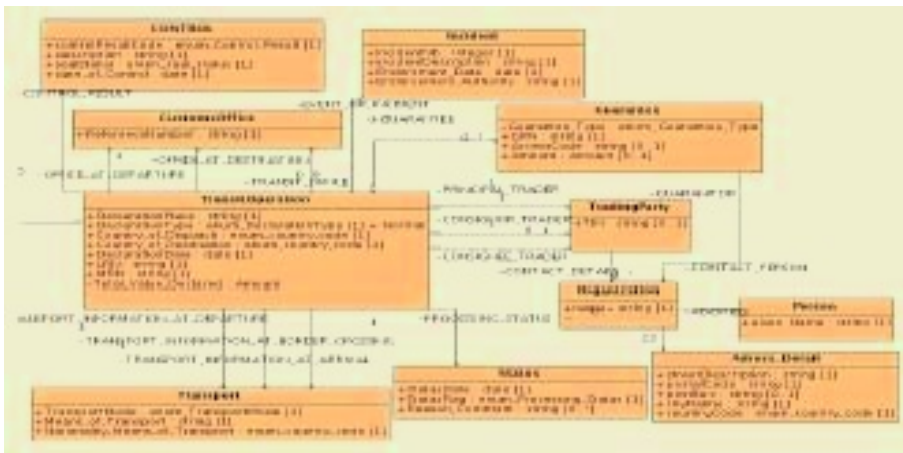


Fig. 2. Part of the NCTS Business Domain

### 3.2 Dynamics of the Transaction: Activity Diagram

Once the basic blocks have been described, the transaction is further specified with more details: the model of a transaction is composed of one activity diagram that describes the flow of business messages among its participants, accompanied by one class diagram for each message exchanged. The activity diagram is composed of:

- Swim-lanes representing the different roles of the transaction
- Activities performed by these roles
- Object flows representing messages exchanged. Each object flow must be preceded and followed by activities belonging to different swim-lanes, these swim-lanes being associated with the role sending and the role receiving the message respectively
- One initial state
- Only one final state (to avoid misunderstandings with workflow formalisms)
- Pseudo-states like forks, joins, decisions and merges (for the same reason, shortcuts on these pseudo-states, like having more than one transitions arriving to or leaving an activity, are forbidden). Only “reported” decisions are supported at the moment: they are decisions without any guard condition, allowing users to choose between 2 or more alternative messages.

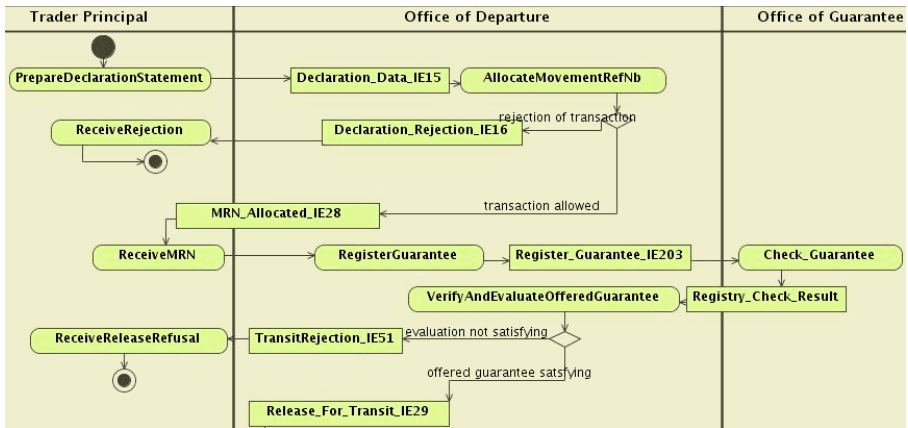


Fig. 3. Beginning of the NCTS transaction

Fig. 3 represents the first part of the NCTS activity diagram: the principal trader submits an export declaration to the Office at Departure, which allows or rejects the transaction. If permission is granted, the Office of Guarantee issues a guarantee and the transaction may go on.

### 3.3 Structure of Messages: Class Diagrams

For each message of the transaction, we define what information it must include depending on the information needs of the recipient at this stage of the transaction. To

do so, we have chosen to use a limited version of UML class diagrams, containing only constructs that can be used to represent XML messages. In order for an XML implementation to be possible, a class diagram must be limited to include only hierarchical relationships, that is, no loops and a unique root class that specifies the root of the message.

Class diagrams we consider contain:

- A “root” class, meaning a class with no navigable relation incoming.
- Classes with attributes of type UML data-types or of another class
- Relations between classes: only binary oriented associations, compositions or aggregations, and generalizations. These relations can have role names, and a multiplicity.
- Possibly additional constructs, like the “enumeration” stereotype, or “XOR” constraints

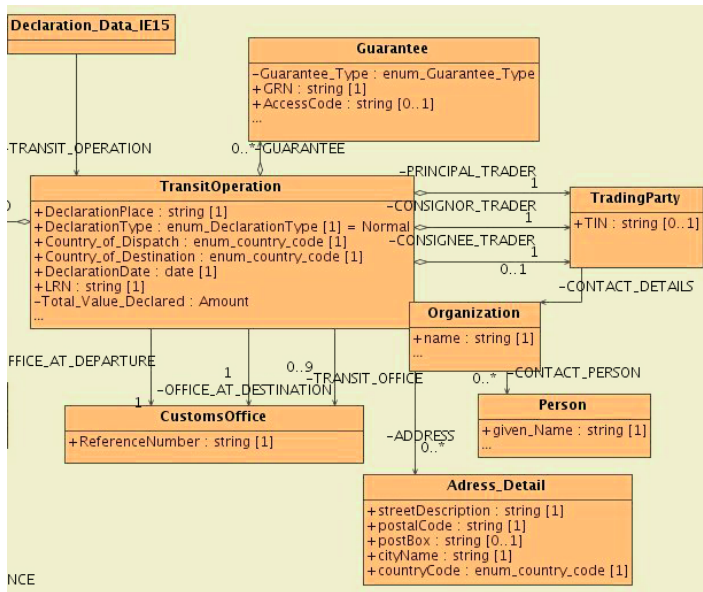


Fig. 4. Part of the class diagram of the “Declaration\_Data\_IE15” message

The class diagrams associated with the different messages are directly built from the global business domain diagram showed in Fig. 2. The information included in the message is a subset of the business domain information.

A part of the class diagram representing the message called “Declaration\_Data\_IE15” of the NCTS transaction is shown on Fig. 4. This message contains information about the transit operation, the type of guarantee, the trading parties, and the cargo to be exported.



## 4 Expressing Business Rules Associated with a Transaction

The Oracle DataBase community has taken over the term "business rule" to designate any data rule than could not be directly represented in their relational database. This table-based rule representation sometimes led to a splitting of one rule in many places of the system. We situate ourselves more in the context of the "logical business rules" community founded by Ron Ross and Terry Moriarty using the name "The Business Rules Group" [15]. Their perspective on business rules is quite similar to what a traditional systems person might call a "system requirement": *trying to formalize what a system should do* (in English from their point of view).

A business rule is part of an accurate description of a transaction. Such a rule states upon specific aspects of this process to lead business participants in doing the right choices.

Formally speaking OCL [12] is the language recommended for expressing rules besides UML diagrams (see e.g. [2]). In practice we have experienced two problems with OCL. On the one hand, the language is not readable at all by business experts, on the other hand the validation of XML documents with regard to such specification is a useless technical challenge. As a conclusion, we have tried to identify the required subset of these rules and choose an ad-hoc formalism for expressing them.

We decided to differentiate 2 kinds of rules: simple, repetitive rules that link the values of elements between different messages, further called inter-message rules, and full-featured business rules allowing the expression of more complex rules.

### 4.1 Inter-message Rules

Inter-message rules are a means to link classes, association ends or attributes of different messages together. They are associated with an element of a class diagram describing a message and specify how the value of this element is related to the content of an element that occurs in a previous message of the current transaction. This kind of rule is especially useful in transactions where most of the data is passed from one message to its successors. Inter-message rules are specified differently from business rules, using simple UML notes, because that makes them much easier and quicker to be added, as we expect that message diagrams contain many such rules.

Some default inter-message rules can be generated automatically by the Efficient plug-in, based on an algorithm that creates a rule each time an element from a previous diagram is re-used with the same attributes and outgoing relations. These default rules can then be modified by the business expert according to his/her needs.

We have differentiated two kinds of rules: the "references" rules express that the value of the referenced element should be recopied, while the "cardinality reference" rules mean that the cardinality of the referenced relation should be preserved. We also have defined optional variants of these rules, which only serve as a help to pre-fill messages but are not enforced. These rules are written as notes on the UML class diagrams, stating the kind of the rule that applies to the linked element, and the "path" (in XML style) of the referenced element on the other diagram. We also have built a small user interface to choose these elements by choosing from a list of classes, associations and attributes.



As you can see on Fig. 5, there are two “reference” rules described by notes containing the name of the rule and the “path” of the referenced element in the “Declaration\_Data\_IE15” message. Here, it means that the Local Reference Number (LRN) of the operation must be the same in the two messages, and also that the Office at departure must be the same. This message also contains other rules not shown on this figure.

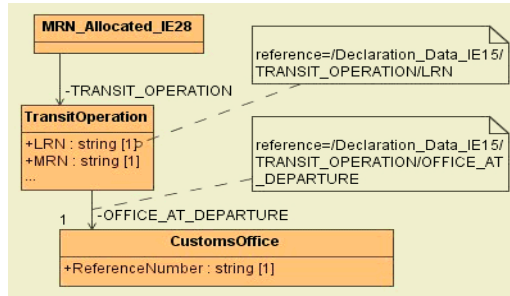


Fig. 5. Inter-message rules on the "MRN\_Allocated\_IE28" message

## 4.2 Complex Business Rules

At the level of the expression of more complex rules, the challenge of choosing a good formalism is twofold: on the one hand the rule needs to be *understandable by business experts*, to help them communicating, on the other hand it should be easily parsed and *understood by computers*. We therefore propose to interpret the same rule in a natural language form (English and French are already supported), and in a computable form (namely Xlinkit, an XML representation of first order logic predicates). The grammatically correct natural language interpretation of the rule is achieved thanks to the open source *Grammatical Framework* (GF) tool of Chalmers University of Technology in Göteborg [7], a strong-typed generic grammar framework.

The "business rules" to be used in the Efficient framework is composed of a *condition* and the (right) *behavior* that responds to such a condition occurrence. The condition describes a situation of the "real life" that may occur during the transaction. We propose to include any factor of the environment that may trigger the transaction as a very part of it, that is: as information provided by some actor of the transaction.

The reaction to a condition occurrence is specified as a particular flow of the transaction, in the UML activity diagram described in Fig. 3, while the way of expressing and computing conditions is detailed below.

### Business Rules Facts and Conditions

For the sake of ease, a rule is being built in a progressive refinement of a tree-like structure, directly mapped onto its natural language meaning to be sure the refinements chosen by the business experts match his intentions. This is done using the graphical user interface of GF [7].

A business rule ranges over *business facts* (see [15]) that represent constructs of the real world in terms of information available during the transaction. Many business facts are derived from the specification of each message, and hence they differ from a transaction to the other. Facts include, among others:

- the simple occurrence of a document
- the exact time at which this document was sent
- any field (element) of the sent document
- any field of the document that is of same kind

From the document of Fig. 4 we propose, for instance, "the name of any Organization" as a fact that describes the name of any of the mentioned traders. The above business facts are combined using typed operators, either common in the business all-day-life as:

- a document or field existence
- a maximal or minimal delay between documents or dates
- a date expiry

or computed using classical arithmetic, boolean and set operators. The case all operators are typed prevents the business expert to refine rules that would not be computable, while the simultaneous natural language interpretation of the rule ensures its semantics.

### Business Rule Context

Technically speaking, a rule may apply under some circumstances (only during a working day, for instance). Those circumstances form what we call the *context* of a rule, users of the Efficient framework may use the facts detailed above to specify a particular context, or simply state that the rule applies either

- always
- after or before a particular document is sent
- between the occurrence of 2 documents (when a rule is triggered *and* expires)

### User Interface

The graphical interface provided by GF displays the current state of the tree-form of the rule, as well as its natural-language interpretation, while giving the ability to further refine the rule by selecting the appropriate element. Here is an example rule, shown in its natural-language form by the tool: "The 'StatusDate is not expired' rule constraints the current transaction after a Declaration\_Data\_IE15 document is sent. *StatusDate of Status in Declaration\_Rejection\_IE16 is not expired.*"

We easily identify the rule context at the beginning of this statement.

### 4.3 Generation of Code for the Business Rules

As explained in Section 2, all the code associated with a transaction is fully and automatically generated from the specification models. Details about the generation of code associated with XML Schemas, workflow procedures and graphical GUI can be found in [5]. It follows according the following principles, that are fully compatible with the ebXML initiative:

- for the structure of messages, we chose to use W3C XML Schema files [24], because XML Schema is a well-known format from the W3C and because there are many open source tools available for data validation and manipulation.
- For the flow of messages, we use a format that is understood by the workflow engine, and that is based on a standard: XPDL (XML Process Definition Language, [25]), from the Workflow Management Coalition [19].

Regarding the code associated with complex business rules, the tree structure that represents a business rule may be interpreted either in English, French, or any other interpretation for which a mapping exists to Xlinkit.

Xlinkit [22] is used to verify consistency constraints that have been defined for a set of XML documents, in our case the XML messages exchanged. Xlinkit implements the most basic logical predicates of the set theory, relying on XPath expressions for the definition of sets of nodes in documents.

### First Order Logic Mapping

The interpretation of a business condition as a first order logic (FOL) predicate is quite straightforward, as illustrated below on the same rule as before.

- a business fact is mapped onto its corresponding XPath expression.
- a condition that may trigger the rule (defined in its context) is translated as a premise of a logical implication operator, whose conclusion is the below operator combination.
- a typed operator is mapped either onto its FOL equivalent if there is one, or onto a function call that implements it.

Xlinkit provides extension mechanisms that allowed us implementing some business-oriented operators (as a date expiration check) and integrating them as common function calls into the animation framework.

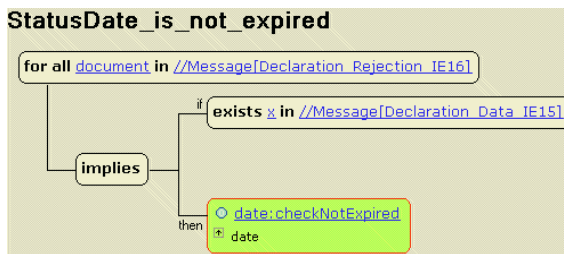


Fig. 6. 1<sup>st</sup> order logic interpretation of a business condition

## 5 Verification and Validation of Transactional Models

The development and the implementation of the Efficient formal modeling language pursues two objectives: the formal representation of the various aspects (flow, content, governance and rules) that impact on an e-business transaction and its

verification. In the context of the Efficient project, the validation of the e-business requirements is achieved by two complementary approaches:

- Automatic verification: the approach based on a proof system aims at verifying that a given property is verified for each possible scenario of the system. . A scenario is associated with one possible execution of the transaction, i.e. one possible path in the activity diagram. In particular, it can be used to verify that the restrictions on the UML language imposed by the Efficient framework are respected in the model, and that there are no logic errors in the model.
- Validation through animation: the animation of transactions aims at discovering errors before the actual implementation of the transaction. Such errors include deadlocks, live locks, missing or non-adequate information content, missing messages or wrong flow of messages, etc. It is the business experts themselves who check the validity of the transaction by visualizing the execution of a set of scenarios that involves the different messages. They simply play the roles of the transaction's actors by receiving messages and sending answers.

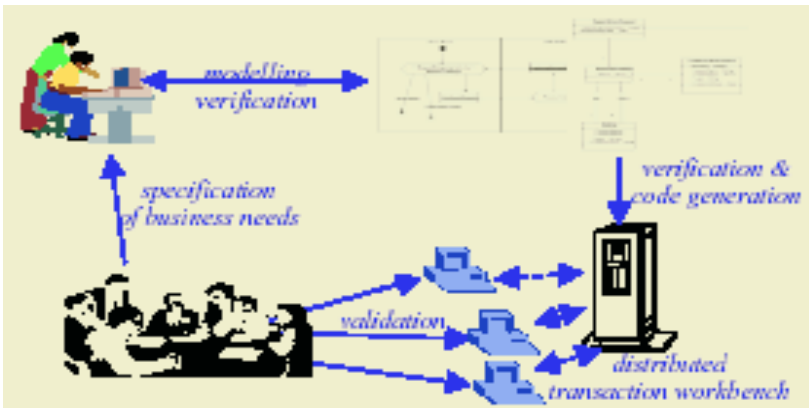


Fig. 7. The Efficient verification and validation activities

### 5.1 Formal Verification

Although the animation approach is well suitable for business experts, its efficiency depends on the relevance of the scenarios chosen. The effectiveness of this approach decreases with the complexity of models that are to be verified: it becomes difficult to find suitable test cases for discovering errors. Besides, this approach can only detect the presence of failures, but cannot prove the correctness of the system we are testing. Contrary to this approach, automatic verification can discover errors that may not be found in the first step because of defects of the test cases. Nevertheless, it is not yet suitable for business experts and thus cannot be used to verify that the model corresponds to their needs.

In the two following subsections, we report about some properties that define a correct e-business process, and then we illustrate their verification approach.

### Properties of Correct e-Business Processes

After studying different kinds of errors discovered during validation by animation, three property categories have been elected:

1. **Structural properties:** this type is closely related to the topology of the diagrams. Structural properties can be directly verified on the UML diagrams without animating the transactions. They are necessary conditions that ensure the feasibility of the transaction. If one of them is not fulfilled, we can assert firmly that the transaction must be erroneous. The structural properties that must be verified by an e-business transaction can be classified into two categories. The first category facilitates the definition of the formal semantics for the class and activity diagrams that make up the transaction. This category comprises the following properties:

- a) A class diagram must not contain circuits. The algorithm used for generating the XPDL code imposes this property.
- b) Each activity diagram must have one initial state and one to several final states. The first part of this property determines the starting point of the transaction; the second specifies its end point. Reaching a final point means that the transaction has been fully accomplished.
- c) The number of incoming/outgoing transitions of each activity is equal to one.

The second category corresponds to general reachability properties. This category includes the following properties:

- a) Each node must be reachable from the initial node. This property ensures that there are no superfluous nodes on a diagram.
- b) From each state, there must exist a path such that one of the final states can be reached. This property ensures that whatever the node reached during the execution of transaction, it may be possible to get to a final state: the system will hence correctly terminate.
- c) To avoid circular dependencies, we assume that for each input node *A* of a join node *C*, there must not exist a path from *C* to *A*. In fact, on the one hand, the fact that *A* is an input node of *C* implies that the node *C* will follow the node *A*, on the other hand, a path from *C* to *A* would imply that the node *A* will follow node *C*: detection of a deadlock.

2. **Real time properties:** these properties aim at evaluating the system in terms of their time responsiveness. Placing constraints on the different activities, we want to be able to determine if the system can answer in a limited amount of time. The different type of time constraints that can be stated on activities include delay, duration, timed events of the form When (t) or After (t), etc.

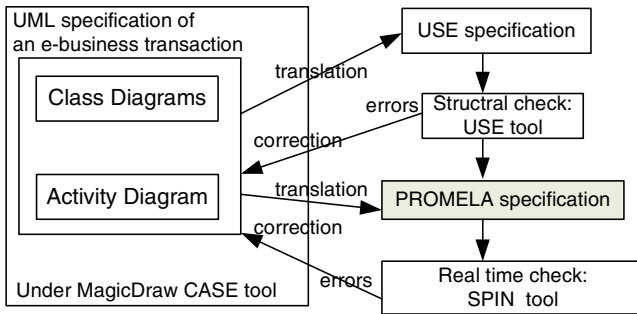
3. **Dynamic properties:** the dynamic feature of these properties means that they are related to two or more states of the system. Its verification is achieved on a set of

states describing a possible evolution of the system. Among the possible set of dynamic properties, we cite the two following safety /liveness properties:

- a) Each guard associated to a transition is not always false (a contradiction). This property is used to eliminate dead paths.
- b) In most domains, applications are desired to be deterministic, that is, at each moment; at most one behavior is possible. For that reason, we impose that the guards must be disjoint.
- c) The disjunction of all the guards yields TRUE. This property is used to eliminate a potential deadlock.

**Our Verification Approach**

The verification process begins by checking the structural properties according to the approach we have defined in [8]. This approach is based on the USE approach fully described in [13]. If these properties are fulfilled, then the real time properties are checked by following a formal approach based on the PROMELA language and its tool support SPIN [10]. More details about this approach can be found in [9]. The reason for checking the structural properties at the beginning of the verification process lies in the fact that these properties are rather simple and fast to verify. Moreover, they allow us to rule out a wide range of ill-defined transactions without getting involved in a deep semantics analysis that may be very time consuming. Fig. 8 depicts the different verification phases.



**Fig. 8.** The verification process

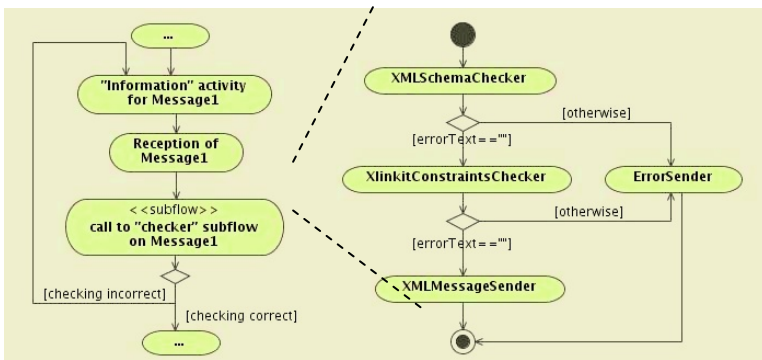
At current time, the verification of structural properties, which is integrated as a Plug-in in the MagicDraw CASE tool, is fully automated. The tool states the number of properties that were verified, including those that were not satisfied, along with the time consumed by the verification task.

**5.2 Validation Through an Animation**

The animator tool orchestrates a process in a distributed scenario: it allows the business users to exchange messages according to the activity diagram of the transaction. When a message is sent, its structure is checked, along with the validity of the business rules specified for the message.

The animator tool is composed of a set of modules, which are integrated around a workflow engine that orchestrates the transaction. The process to be executed by the workflow engine is based on the process modeled in the activity diagram, enriched with the following additional activities (see Fig. 9):

- Before a message can be sent by a role, an additional activity is introduced informing the different participants about the messages that they can send.
- The reception activity is called when the message is received. It stores the XML message received in a XML database
- Then the structure of the message is checked as well as the associated business rules, in a “check” sub-process (see Checker sub-flow on Fig. 9).
- If the checking returns an error, an error message is sent to its sender
- Otherwise, the message is forwarded to its recipients (with eventually pre-filled values, depending on the inter-message rules associated with it)



**Fig. 9.** Process for each message in the animator, and checker sub-flow

Efficient provides the recipients with a web based client tool that allows them to:

- Instantiate one of the processes for which the animator is configured, by identifying potential participants along with the roles they play in the transaction
- Receive messages addressed to one of the roles they play (using web services)
- Send messages, after the animator has executed a corresponding “information” activity, by using web-based forms to input data and web services to send it. The forms are XForms files [21], and are displayed using an open source XForms implementation called Chiba [1]
- See the list of previously received messages

In general, in the animator tool and in the client, we have favored XML standards over other formats, as well as the use of open source tools over proprietary tools. For example, the animator uses an open source workflow engine called WFMOpen [20], and use different XML libraries to parse and validate XML files.

## 6 Conclusion

This article introduces the Efficient toolset, a comprehensive environment for designing and evaluating e-business transaction. Efficient facilitates the communication between business experts and IT analysts and allows them to progressively elaborate the specification by generating a prototype of what the transaction might look like, without having to implement it. Current and future work will focus on the enhancement of the expressiveness of the modeling language: choices between two or more activities may be e.g. based on the actual values of a message form, a more flexible definition of roles will be investigated on, and we are going to evaluate the use of time constraints. Also, an upgrade to UML 2.0 will be considered. From the verification point of view, we plan to extend the developed verifier tool to take real-time features into account. Another interesting direction is to consider nested transactions. The concept of “nested transactions” allowing a specification reuse, it would be interesting to study how the correctness of a nested transaction can be established (or deduced) from the correctness of the sub-transactions that compose it.

## References

1. Chiba, open source Xforms implementation, <http://chiba.sourceforge.net>
2. Corr ea et C. Werner, *Precise Specification and Validation of Transactional Business Software*, in Proc. of IEEE Joint International Requirements Engineering Conference (RE'05), Kyoto, Japan, 2005.
3. ebXML, <http://www.ebxml.org/>
4. R. Eshuis, P. Brimont, E. Dubois, B. Gr egoire, S. Ramel, *EFFICIENT: A Tool Set for Supporting the Modelling and Validation of ebXML Transactions*, poster and short paper at ESEC/FSE 2003, <http://efficient.citi.tudor.lu/>
5. R. Eshuis, P. Brimont, E. Dubois, B. Gr egoire, S. Ramel. *Animating ebXML Transactions with a Workflow Engine*, In Proc. CoopIS 2003, volume 2888 of LNCS, Springer, 2003.
6. extensible Markup Language, <http://www.w3.org/TR/2004/REC-xml-20040204/>
7. *Grammatical Framework*, open source tool, <http://www.cs.chalmers.se/~aarne/GF/>
8. N. Guelfi, A. Mammam, B. Ries, *A Formal Approach for the Specification and the Verification of UML Structural Properties: Application to E-Business Domain*, International Workshop on Software Verification and Validation (SVV 2004), workshop of ICFEM'04, Seattle, WA, USA, 2004.
9. N. Guelfi, A. Mammam. *A Formal Approach for the Verification of E-Business Processes Using the PROMELA Language*. Submitted to FASE'05.
10. G.J. Holzmann, *The Model Checker SPIN*, Software Engineering 23(5)pp.279-95, 1997.
11. MagicDraw, commercial tool, <http://www.magicdraw.com/>
12. Object Constraint Language (OCL), <http://www.omg.org/cgi-bin/doc?formal/03-03-13>
13. M. Richters, *A Precise Approach to Validating UML Models and OCL Constraints*, PhD Thesis, University of Bremen, 2001.
14. RosettaNet Partner Interface Program (PIP), <http://www.rosettanel.org>
15. R. Ross, *The Business Rule Book: Classifying, Defining and Modeling Rules*, Second Edition, 1997.



16. M. Schmitt, B. Grégoire, C. Incoul, S. Ramel, P. Brimont and E. Dubois, *If business models could speak! Efficient: a framework for appraisal, design and simulation of electronic business transactions*, ICEIMT 04, <http://efficient.citi.tudor.lu>
17. UN/CEFACT Modeling Methodology (UMM), <http://www.ebxml.eu.org/umm.htm>
18. Unified Modeling Language, Object Management Group specification, <http://www.omg.org/uml>
19. WfMC, Workflow Management Coalition, <http://www.wfmc.org/>
20. WFMOpen, open source workflow engine, <http://wfmopen.sourceforge.net/>
21. Xforms, W3C standard, <http://www.w3.org/TR/2003/REC-xforms-20031014/>
22. *Xlinkit*, commercial tool, <http://www.systemwire.com/xlinkit/>
23. XML Common Business Library (xCBL), <http://xml.coverpages.org/cbl.html>
24. XMLSchema, W3C standard, <http://www.w3.org/XML/Schema>
25. XPDL, standard from the WfMC <http://www.wfmc.org/standards/XPDL.htm>,
26. W.M.P. van der Aalst, H.M.W. Verbeek, and A. Kumar, *XRL/Woflan: Verification and Extensibility of an XML/Petri-net based language for interorganizational workflows*, <http://tmitwww.tm.tue.nl/staff/wvdaalst/publications/p139.pdf>
27. Nick Szirbik, Gerd Wagner, *Steps Towards Formal Verification of Agent-based E-Business Applications*, <http://www.informatik.uni-hamburg.de/TGI/events/moca01/wagner-final.pdf>

# Separation of Structural Concerns in Physical Hypermedia Models

Silvia Gordillo<sup>1,3</sup>, Gustavo Rossi<sup>1,4,\*\*</sup>, and Daniel Schwabe<sup>2,††</sup>

<sup>1</sup> LIFIA, Facultad de Informática, UNLP, La Plata, Argentina  
{gordillo, Gustavo}@sol.info.unlp.edu.ar  
<http://www-lifia.info.unlp.edu.ar>

<sup>2</sup> Departamento de Informática, PUC-Rio, Rio de Janeiro, Brasil  
dschwabe@inf.puc-rio.br  
<http://www.inf.puc-rio.br>

<sup>3</sup> Also CICPBA

<sup>4</sup> Also CONICET

**Abstract.** In this paper we propose a modeling and design approach for building physical hypermedia applications, i.e. those mobile applications in which physical and digital objects are related and explored using the hypermedia paradigm. We show that by separating the geographical and domain concerns we gain in modularity, and evolution ease. We first review the state of the art of this kind of software systems, arguing about the need of a systematic modeling approach; we next present a light extension to the OOHDM design approach, incorporating physical objects and “walkable” links; next we generalize our approach and show how to improve concern separation and integration in hypermedia design models. We compare our approach with others in the field of physical and ubiquitous hypermedia and in the more generic software engineering field. Some concluding remarks and further work are finally presented.

## 1 Introduction

The idea of physical hypermedia (PH) was first introduced in [7] to support design activities and to organize collections of different media, and in [15] as a formalism to build augmented reality applications. In these software systems, physical objects are augmented with digital information which can be accessed by the mobile user, for example while standing in front of the object. Physical objects can be further considered as nodes in a hypermedia network and thus linked with other nodes either physical or digital. When dealing with digital objects, the user will traverse the link using the well-known navigation paradigm (e.g. as in the WWW); in other cases (e.g. when physical objects are involved) the link will have to be “walked” by the user [9].

---

\*\* Gustavo Rossi is partially funded by Universidad Abierta Interamericana in its project “Conceptual Modelling of Web Applications”.

†† Daniel Schwabe is partially funded by CNPq – Brasil.

A simple example scenario is a museum in which visitors are equipped with portables computer devices. When the visitor stands in front of an artwork, he gets multi media information about the artwork in his portable device. Additionally, he is presented with a set of anchors that allow him to navigate to other objects (hypermedia nodes) related with the artwork. When one of these nodes is another artwork in the museum, he can be shown how to reach this artwork; he can then choose to traverse the physical space (walk the link) towards this node, or just continue his actual tour. Notice that we are not just augmenting the physical object (artwork) with some digital information but also providing some kind of linking to other digital or physical objects.

These ideas allow to apply the hypermedia paradigm to the real world; it has been shown elsewhere [5] that we can also build rich social interactions when users can link their own comments to a physical object, for example as digital graffiti or recommendations, and these comments can be accessed or further discussed by other users.

In our research we are interested in how to model and design these applications, i.e. in analyzing which software modeling and design issues we face while building PH applications, which software abstractions we need to clearly indicate the intended structure and behavior of a PH network, and the way in which those abstractions relate with each other.

In this paper we present a novel approach for the design of PH applications; this approach seamlessly extends the Object-Oriented Hypermedia Design Method (OOHDM) [18] to support physical objects and “walking” navigation. We show that, by clearly separating the fundamental concerns in this kind of software, we improve modularity and ease of evolution. We also show how to go further with this approach in complex application domains, by applying well-known techniques for separation of orthogonal (or partially overlapping) concerns.

The main contributions of this paper are the following:

- We indicate which design issues must be faced while modeling PH applications,
- We present a concise design approach that can be easily adapted by other hypermedia and Web modeling methodologies like UWE [11] or WebML [3] to extend their scope to the physical world.
- We present an approach for designing navigational structures that depend on application concerns. We define concern-driven navigation as an extension of our approach of concern decoupling in physical hypermedia.

The rest of the paper is organized as follows: In Section 2 we introduce our modeling approach; we briefly explain the OOHDM framework and indicate how we extended it to support PH and we analyze several navigation issues. In Section 3 we discuss how to generalize our basic approach to include other concerns. In Section 4 we compare our work with others both in the hypermedia and software engineering fields. Some further work and concluding remarks are finally described in Section 5.

## 2 Modeling Physical Hypermedia Applications

While researchers have emphasized the feasibility of the PH paradigm by building software infrastructures that support these ideas [8,9,15] and performing usability studies [7], modeling and design issues have been so far ignored. We believe that the inherent complexity of these applications requires a special emphasis in modeling and design.

To make this discussion concrete, we define a PH application as a hypermedia application (i.e. the access to information objects is done by navigation) in which all or some of the objects of interest are real-world objects which are visited by the user “physically”. The most usual scenario for these applications involves a mobile user and some location sensing mechanism and underlying software that can determine for example when the user is within interaction range of one of these objects.

In this specific domain we need to express, in an implementation-independent way, which are the objects of interest and their properties (including their location), how they are linked, which links should be implemented as conventional and which should be “walked” by the user. We should be able to cope with technology evolution and heterogeneity, i.e. the design model should not be compromised with details on location-sensing technology and at the same time it should allow to build models that can gracefully evolve together with new technical possibilities.

We have extended the OOHDM [18] design approach by adding a few concepts such as physical objects and slightly changing some navigation semantics to adapt them to the physical hypermedia field. The object-oriented nature of OOHDM and its open meta-model allowed us to achieve this objective easily, without changing the basic assumptions and primitives of the methodology. In the following sub-sections we stress those modeling constructs that are fundamental to the development of this kind of context-aware software. In particular we emphasize how to describe basic structures and behaviors in a high level way. We purposely ignore aspects related with user modeling and user context-aware adaptation that have been discussed elsewhere [1,10,19].

## 2.1 The OOHDM Design Approach

OOHDM partitions the development space into five activities: requirements gathering, conceptual design, navigation design, abstract interface design and implementation. The first step is to elicit stakeholders’ requirements which helps to identify actors and the tasks they must perform. Scenarios are collected by means of User Interaction Diagrams, a special form of Use Cases. During conceptual design we describe the application classes and their relationships using UML [21].

For each user profile we can define a different navigational structure according to the tasks this kind of user must perform. The linking structure of a Web application is then defined by a navigational schema, built from navigational meta-classes such as nodes, links, anchors and access structures such as indexes. Each node is defined as a view over conceptual objects, acting as an Observer on those objects [6]. The separation between objects and their views allows customizing the structure of nodes and the linking topology to the needs of the corresponding user profile and task.

Additionally, the navigational contexts schema defines the meaningful sets of nodes that the user will traverse and the intra-set navigation topology. For example, we can specify the navigational context “Artworks by Painter” to denote the set of paintings of a particular painter, and specify that sequential navigation (from one artwork to the next, according to some specified ordering) is allowed. Since the same artwork might belong to different sets, e.g. Artworks in a Time Period or Artworks in a Room, the differences when accessing it in one context or others, are expressed using InContext classes, that extend the basic features of a node in the particular navigational context. Details on these primitives can be found in [19].

The abstract interface model defines which interface objects the user will perceive (in particular how nodes will look like) and which interface transformations will take place. Finally, during the implementation activity the whole set of models is mapped into a run-time environment. Though OOHDM does not prescribe a particular strategy for implementing a hypermedia or Web application, its approach can be naturally mapped to object-oriented languages and architectural styles, such as the Model-View-Controller. In the following sub-sections we concentrate in the conceptual and navigation design activities.

## 2.2 Dealing with Physical Objects

We extended the OOHDM conceptual meta-model by adding the concept of Physical Object and a simple User Model. Though the user model is not described here for the sake of conciseness, it suffices to say that it contains information about the current user's position together with his (general or application specific) preferences. Details on the introduction of user's information in OOHDM models are discussed in [19].

A physical object is an application object that can be explored "physically", i.e. it has a physical presence in the system and the user can be tracked if he is within interaction range of it. In the museum example, we can be interested in modeling artworks, rooms, corridors or other places as physical objects.

To find a suitable approach for modeling physical objects, we need to consider that not all objects of the same class (e.g. Artwork) should be tagged as physical: for example, in the museum, we might want to relate artworks in the museum with others that are not in exhibition, or are in another geographical place, or simply do not exist anymore. The "physicality" of an object is a completely separate concern from its other characteristics, such as its (sub) type. Therefore, representing physical objects as sub-classes of a particular class (e.g. Artwork) introduces a specialization criteria that might collide with others in the intended domain (paintings, sculptures, etc) and also prevents us for considering an artwork alternatively as physically accessible or not for when it is not longer in the museum or when it is presented in an exhibition. To accommodate this, we have then chosen to model physical objects as roles that can be assumed by conceptual objects. Roles have been extensively used to model and integrate different points of views on the same reality [13], both as conceptual modeling and design artifacts [14,20]. Roles can easily be mapped to simple implementations either using decorators as shown in [14] or even Java interfaces.

A role type (in this case a sub-type of the basic role Physical) indicates those properties and behaviors of an object when playing that role, i.e. when the object has a physical presence. In Fig. 1 we show a simple conceptual model for the physical museum; roles are described using the notation in [14]. Small white boxes indicate that the corresponding class can play the corresponding physical role, i.e. any object of the class can be considered physical.

Physical objects are characterized by an attribute *location* whose semantics depends on the location model being used, and an operation to change location (if the object is mobile or can be changed of place); *location* can be just an identifier (e.g. if we use code bars or infrared sensing), or we might need a more complex representation. In our meta-model we provide a set of basic geometries and reference systems

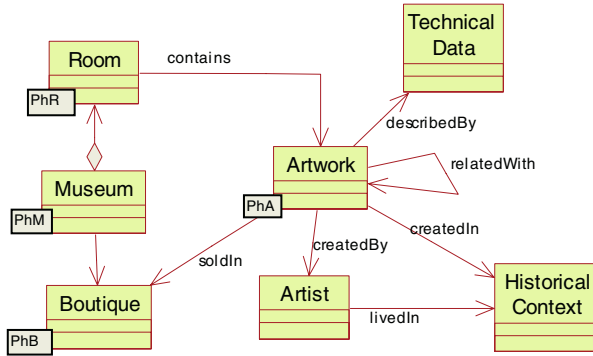


Fig. 1. Conceptual Model of the museum including physical objects

for locations as shown in Fig. 2. A designer might choose one of them or define his own location model. Using the simple solution of Fig. 2, we can deal with different representations and location-sensing technologies even for the same class of objects; for example if an artwork is located in a square outside the museum we could use global coordinates, as usual with GPS technology. This design structure also simplifies evolution when location or sensing technology changes, as the design model can be seamlessly adapted to the evolution, for example by adding a new type of reference system.

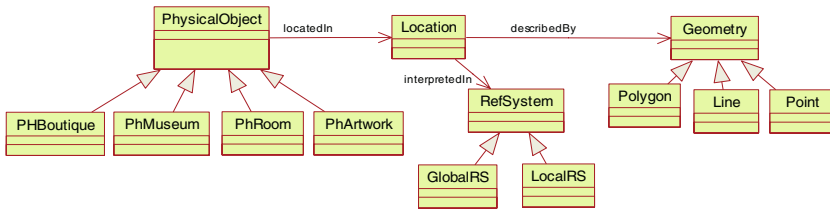
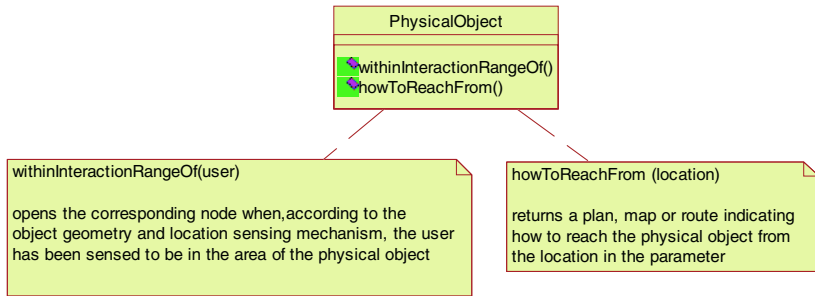


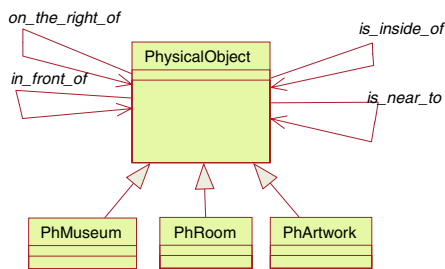
Fig. 2. Decoupling Location model from physical objects

The location of a physical object involves a value expressed in a Geometry (point, line, polygon, etc.) and interpreted in one reference system. For the sake of simplicity we do not explain details of location interpretations, which has been well covered in the literature of geographic information systems [12].

Physical (role) objects possess a default behavior that allows them to handle the event signaled by the user being within interaction range, by opening (activating) the corresponding node. Additionally, they could be able to inform how the user can reach them from any location; this behavior is triggered by “walkable” links (See 2.3) to indicate how the user can “navigate” physically to the object. The object can either answer its absolute location, a plan or the route one must follow to reach it from the actual location. The designer must specify this last and eventually other behaviors as they are application dependent. In Fig. 3 we schematize the fundamental spatial behaviors.



**Fig. 3.** Spatial Behaviors of Physical objects



**Fig. 4.** Relationships among Physical Objects

Physical objects may be related using a variety of spatial and geographical relationships and predicates which give us a tool to build interesting navigational structures as will be described in Section 2.3. In Fig. 4 we show some well-known spatial relationships. Notice that their implementation in a particular system might need that they are specified in an instance basis (e.g. an artwork *in\_front\_of* another). In other cases they can be calculated using the corresponding geometrical behaviors [12]. Designers can devise new relationships specific to their domain of interest. These relationships will be shown in the model of Fig. 1 as relationships between the physical roles of the corresponding classes.

### 2.3 Navigation Issues

Navigation in PH applications is also described by a navigational schema with the OOHDM semantics. This allows us to eventually let a user explore a physical object even when not physically near it, by just defining the corresponding Node class; in our example this is useful for those artworks under restoration, which will be only accessed physically by specialists.

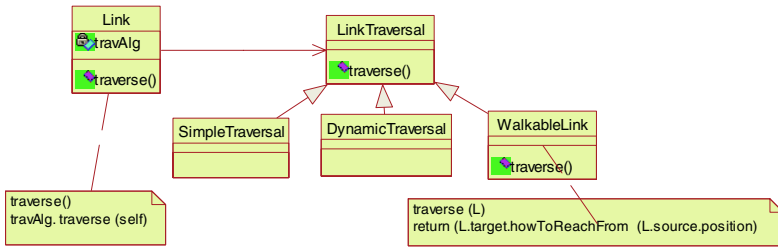
There are, however, two major differences between a conventional and a physical hypermedia regarding the operational semantics expressed by the navigational schema: the activation of nodes and the semantics of link traversal.

In conventional hypermedia a node is opened when we navigate a link having that node as a target. While we want to preserve this behavior for “pure” digital nodes, a node that stands for a physical object should be opened when the user is within interaction range of the object. Of course we might also want to build a “conventional” web interface for that object which can be trivially done using the viewing mechanisms of OOHDM, so we won’t discuss this possibility here.

We decided not change the basic Node class hierarchy; as we explained in Section 2.2, a node corresponding to a physical object will be opened when the user is within interaction range of the object as a consequence of the default behavior specified for its role as a physical object. We can of course implement more sophisticated context-aware or personalized activating behaviors; this can be done by using the strategies discussed in [19].

Meanwhile, to implement a different navigation semantics, we defined “walkable” links (or WLinks) as those links whose target node is the digital counterpart of a physical object. The main difference between the operational semantics of a navigational and a walkable link is that, while the former usually closes the current node and opens the target node, the latter just indicates the user intention to reach the corresponding physical object.

WLinks are designed by changing the default link traversal algorithm, which is expressed in OOHDM as a Strategy [6] on Link classes as described in [18] and shown in the left of Fig. 5. In order to achieve the desired behavior, the link traversal algorithm invokes the *howToReachFrom* behavior (Fig. 3) in the physical object corresponding to the target node; the actual user location used as a parameter is the location of the link source node. A schema of the decision structure of the WalkingLink traversal is shown in Fig. 5.



**Fig. 5.** Walkable links as Strategies on Links

This design style allows implementing different types of “walking” semantics by specifying a different algorithm as another Strategy class, either at the same level of *WalkableLink* or as a concrete sub-class that re-writes the method *traverse*. For example, in a production line application, we might want that the target object moves towards the user, so instead of asking for a map we could send a message such as *L.target moveTo(L.source.position)*.

Decoupling links from their traversal algorithms also allows us to express differences at the link instance level, for example when an instance of a *WLink* class has an exceptional “non-walking” semantic, i.e. it behaves as a conventional link, and we



show digital information of the object even if the user is not in front of it. In Fig. 6 we show the navigational schema for the visitor user role that corresponds to the conceptual model in Fig. 1. WLinks are shown with a `<<W>>` in the style of UML stereotypes [21]. We omit the name of link classes for the sake of simplicity. With this solution we also cope with evolution problems that are mapped to changes in instances of the corresponding Link class instead of requiring changes in the overall class.

WLinks, in the same way as navigation links, allow us to explore physical objects by mapping conceptual relationships shown in Fig. 1 into walkable links. For example two artworks that are closely related might be linked to suggest the user a less conventional museum visit. Notice that additional usability issues arise, e.g. to avoid suggesting the user to perform long walks: they also become a design concern. Interesting discussions on this subject can be found in [7,8].

Physical objects can also be related through rich spatial relationships either generic as shown in Fig. 4 (e.g. near, in front of) or application specific (e.g. in the same room), that may also induce interesting navigation relationships and structures. These relationships can help to create navigational contexts by grouping objects according some location property; for example we can describe the set of Rooms close to the Boutique, or the set of Artworks located not far from a certain zone in the Museum. In a tourist application we might want to visit the monuments along a road or a river, or those that are near a particular place. It is interesting to note that, when using WLinks to implement intra-set navigation in a navigational context, the context represents a real guided tour along a geographical space. Furthermore, these location-based relations can be combined with the other types of relations in defining meaningful contexts; for example, while traveling in southern France, one may wish to see the Artworks by Van Gogh depicting scenes in that region.

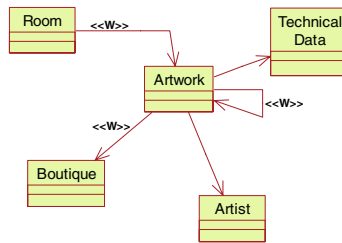


Fig. 6. Navigational Schema with WLinks

### 3 Advanced Concern Separation

In previous sections we showed how to decouple conceptual properties from spatial properties of physical objects. We have also shown that the spatial perspective opens new yet unexplored possibilities for building navigation structures. While the spatial concern is crucial for physical hypermedia, we can generalize this discussion to other non-spatial concerns. Some of them will be orthogonal with the others: for example, the user model could be described without directly affecting application classes [19].

Other types of concerns however are more difficult to handle. Suppose that in the Museum example we want to model the history of the museum, e.g. how the building and collections evolved over time, when artworks arrived at the museum, and why. Should we clutter the existing model (e.g. Fig. 1) with information and relationships related to this new concern? Additionally, how do we deal with the cognitive overhead eventually produced by links belonging to different “themes”? We introduce these ideas in the following sub-sections by first formalizing how to clearly separate spatial from conceptual modeling.

### 3.1 Conceptual Versus Spatial Modeling

By using well known techniques for concern separation such as Role Modeling [16] or Subject-Oriented Design [4], we can re-think the conceptual model of Fig. 1, as dealing with two separate sub-models and specify them as shown in Fig. 7, with UML packages, namely *KernelMuseum* and *PhysicalMuseum*.

Thus, each model can be engineered separately, and relationships can be thought and evolve independently thus improving modularity. When two classes with the same name exist in both models (e.g. Room, Artwork, Boutique) we need to solve this overlap when integrating the models. We did this as shown in Fig. 1 by using roles in classes with the same name. Fig. 1 therefore represents a design refinement of the higher level model in Fig. 7.

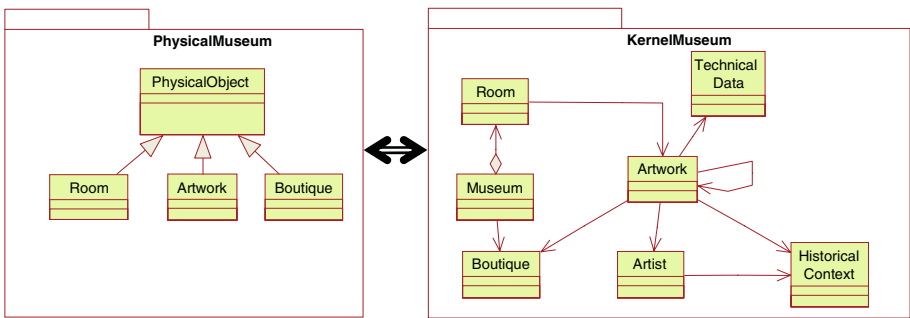
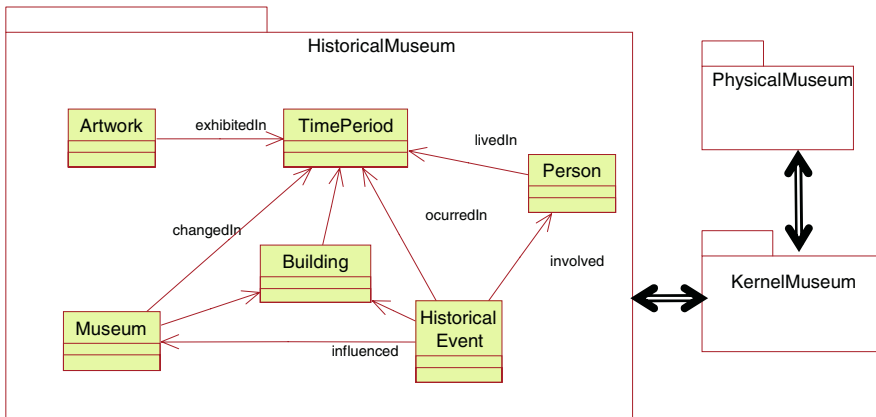


Fig. 7. Decoupling design concerns

Notice that the *PhysicalMuseum* model might also contain relationships between application classes, that will be mapped onto relationships between the corresponding (physical) role classes.

### 3.2 Extending the Approach to Other Concerns

We can use the same design philosophy shown in Fig. 7 when introducing other concerns into the conceptual design activity. Generalizing the ideas behind Fig. 7, we show in Fig. 8 a model that adds the *Historical* concern of the Museum to the previous example.



**Fig. 8.** Multiple concerns in conceptual models

In Fig. 8 we considered the *KernelMuseum* concern as the central one; historical and physical classes and relationships should be further integrated with those in the kernel. This can be done again using roles in overlapping classes; in this example we will have a *Historical* role for *Artwork* and *Museum*; each role will host attributes and relationships that belong to the corresponding concern. Interesting problems arise when dealing with more complex applications or behavioral requirements (see for example [4]). Other techniques such as Subject-Oriented Design [4] or Aspect Oriented Design [2] can be used to describe the conceptual model when cross-cutting requirements exist, though they are usually applied to more technical or programmatic concerns as discussed in section 4.

### 3.3 Discussion: Towards Concern-Driven Navigation

It is interesting to analyze the impact of introducing multiple concerns in the navigational model; in our example, what does it mean to navigate the hypermedia space following one of those concerns? What kind of software abstractions do we need in order to support this kind of navigation?

We say that a hypermedia application supports concern-driven navigation, when it is possible to choose one particular application concern and emphasize the information and links corresponding to that concern, even at the expense of eliminating others. For example, a user exploring the *Historical* concern of the museum might want to ignore all information not related with this concern, or perhaps be able to “switch” from one concern to another. Meanwhile, a person on a tourist trip might just want to be “pointed” to places of interest, instead of just navigating through digital data. Some concerns might be mutually exclusive, while others might be pervasive; e.g. in the physical museum, the spatial concern is always accessible, since the user is traversing the museum physically.

The discussion in 3.2 also applies to the navigation model; a designer might derive different navigational schemas, one for each possible concern, using OOHDM primitives, and then integrate them in a single schema, once again using roles. In the role

model, while the base node class specifies information that the nodes always exhibit, a role type will indicate which additional information and links a node will show when playing the corresponding role, i.e. when accessed within the corresponding concern. Navigational role types are derived from those roles representing concerns in the conceptual model. It is a designer choice to decide which navigation concerns are useful for a particular user profile or task, e.g. using the OOHDM viewing mechanism for a specific hypermedia application. A similar solution has been used in [17] for solving more general navigation problems.

We do not impose a particular kind of association among roles and nodes. Whereas the standard unidirectional association is taken as a default (roles know about base nodes but not vice versa), a designer might want to make nodes aware of the roles it can play, in order to provide additional navigation operations. He can design an operation for changing the actual concern (role) by another to allow more flexibility in the user navigation; this can be useful, for example, for adaptive hypermedia. For example, while exploring one node from a historical point of view, we might want to see the “other face” of the node and continue exploring the physical concern. The idea of concern-driven navigation opens many additional design issues, outside the scope of this paper.

## 4 Related Work

Some of the design problems addressed in this paper have been the focus of interesting research projects in the hypertext and object-oriented communities. We next discuss some of them to highlight our contributions.

### 4.1 Hypermedia

In [8] a comprehensive framework (HyCon) for deploying applications in which the hypermedia paradigm is extended to the physical world is presented. The authors not only show how to provide situated authoring and browsing but also show different usage patterns in this kind of applications. In [15] meanwhile, an object-oriented framework called HyperReal, based on the Dexter hypertext reference model is presented. As in [8] the authors show a powerful software substrate for building augmented reality software. Our research is more oriented towards the modeling and design of physical hypermedia applications: once the intended structure and behavior of a PH application has been specified using the extended OOHDM, it could be implemented for example using HyCon or HyperReal. We believe, however, that for these technologies to become mainstream, some standardization at the level of implementation architectures is needed. We are now studying how to extend the MVC metaphor to include the physical dimension (See Section 5).

The goal of Adaptive Hypermedia (AH) [1] has been to improve the usability of hypermedia applications by taking into account user interests and profiles. In this way adaptive hypermedia has studied how to adapt the contents (nodes) and topology (links) of the application according to a user model. Mature research in AH has led to separating the user model from adaptation rules and from the domain model. The UWA project [10] meanwhile has dealt with providing ubiquitous access to Web

applications. A comprehensive modeling and design approach including user models and adaptation rules has been devised.

Although PH is a kind of ubiquitous and adaptive hypermedia software, our research has a different intent with respect to providing adaptive behaviors. First, we are studying how to design applications in which real and digital objects are linked with the hypermedia paradigm, and exploring how to build meaningful navigation structures that take into account the spatial domain. Besides, we are carrying separation one step further than in UWA and AH by applying it to specific application concerns, e.g. the physical concern.

## 4.2 Object-Oriented Modeling and Design

Separation of concerns has been a recurrent theme in the software engineering and in particular the object-oriented field. Many researchers have argued that the object abstraction is not enough to solve problems such as cross-cutting concerns, misalignment between requirements and designs and evolving behaviors. These problems have been addressed using Aspect-Oriented Programming [2], Subject-Oriented Programming and Design [4] and Role Modeling [16]. Aspect-orientation has focused mainly on technical domains, such as persistence, caching, security, etc. Subject-Oriented Programming has been first used at the programming level and more recently for aligning requirements with designs.

Our work is grounded on the ideas of Role Modeling (in fact we use the role construct heavily in our approach), but with an original application focus: to separate physical from more conceptual aspects. Subject and Aspect Orientation have not been used in the field of hypermedia so far. Our concept of concern-driven navigation meanwhile has not been addressed previously in the literature, although the original OOHDM InContext class primitive can be seen as a first step in this direction.

## 5 Concluding Remarks and Further Work

In this paper we have presented an original approach for modeling physical hypermedia applications, i.e. those applications in which physical and digital objects are related using the hypermedia paradigm. We have shown how to extend our approach to a broader domain: to build hypermedia applications in which there are many different concerns, for example corresponding to application “themes” or subjects. For space reasons we have not discussed customization and personalization issues; given that PH applications are a particular example of context-aware software, these issues are fundamental. It is relatively straightforward to apply previous ideas on Web applications customization [19] in this extended OOHDM framework.

We are currently working on several research directions. One of them relates with providing better modeling tools to express navigational structures. Physical navigation introduces a new kind of context, different from the OOHDM idea of navigational context: the actual location of the user is relevant to provide him (physical) navigation cues, for example in the form of links or landmarks. Suppose that the user is in front of an object of interest (e.g. an artwork); the corresponding node exhibits two kind of links: navigational and Wlinks, the latter ones allow navigation to other

physical objects in the environment. A reasonable design decision could be to keep those links active (visible) while the user navigates to other digital objects related with the artwork (as he is still in the same physical position). In this way, he is always aware of the actual physical navigation options he has from this location. This is certainly a navigational design problem; we want to express that some of the nodes reachable from one particular node (the one corresponding to a physical object) “inherit” the walkable links of this node. We are studying how to solve this design requirement by using a slight modification of the OOHDM concept of composite nodes.

We are also improving our notation to make it more “standard” by exploring the use of UML stereotypes and the OCL [21] to express design constraints. We are also researching on the exploitation of the geo/spatial dimension in order to find a better way to integrate it into other design concerns. We are finally working on implementation issues, such as adapting the MVC architectural style to physical hypermedia applications. In the context of a prototype implementation for the Museum of Natural Sciences in La Plata we are also experiencing usability aspects and their relationships with the concepts presented in this paper.

## References

1. Adaptive Hypermedia Home Page: <http://www.wis.win.tue.nl/ah/>
2. Special Issue on Aspect Oriented Programming. *Comm ACM*, October 2001.
3. Ceri, P., Fraternali, P.: Web Modeling Language (WebML): a modeling language for designing web sites. *Computer Networks and ISDN Systems*, 33(1-6), June (2000) 137-157
4. Clarke, S.: Composition of Object-Oriented Software Design Models. Ph.D. Thesis, January 2001, Dublin City University. In: [www.cs.tcd.ie/Siobhan.Clarke/papers/SClarkeThesis.pdf](http://www.cs.tcd.ie/Siobhan.Clarke/papers/SClarkeThesis.pdf)
5. Espinoza, F., Persson, P., Sandin, A., Nystrom, H., Cacciatore, E., Bylund, M.: GeoNotes: Social and Navigational Aspects of Location-Based Information Systems”. *Proceedings of Third International Conference on Ubiquitous Computing (UbiComp 2001)*, Springer Verlag, 2-17
6. Gamma, E., Helm, R., Johnson, J., Vlissides, J.: Design Patterns. Elements of reusable object-oriented software, Addison Wesley 1995
7. Gronbaek, K., Kristensen, J., Eriksen, M.: Physical Hypermedia: Organizing Collections of Mixed Physical and Digital Material. *Proceedings of the 14<sup>th</sup>. ACM International Conference of Hypertext and Hypermedia (Hypertext 2003)*, ACM Press, 10-19
8. Hansen, F., Bouvin, N., Christensen, B., Gronbaek, K., Pedersen, T., Gagach, J.: Integrating the Web and the World: Contextual Trails on the Move. *Proceedings of the 15<sup>th</sup>. ACM International Conference of Hypertext and Hypermedia (Hypertext 2004)*, ACM Press, 2004
9. Harper, S., Goble, C., Pettitt, S.: proximity: Walking the Link. In *Journal of Digital Information*, Volume 5, Issue 1, Article No 236, 2004-04-07. Available at: <http://jodi.ecs.soton.ac.uk/Articles/v05/i01/Harper/>
10. Kappel, G., Proll, B., Retschitzegger, W.: Customization of Ubiquitous Web Applications. A comparison of approaches. *International Journal of Web Engineering and Technology*, Inderscience Publishers, January 2003
11. Koch, N., Kraus, A.: The authoring process of UML-based Web Engineering Approach. In *Proceedings of the 1<sup>st</sup> International Workshop on Web-Oriented Software Construction (IWWOST 02)*, Valencia, Spain (2001) 105-119

12. Laurini, R., Thompson, D.: *Fundamentals of Spatial Information Systems*, Academic Press Ltd, 1992
13. Pernici, B.: Objects with Roles. *Proceedings of the ACM-IEEE Conference on Office Information Systems (1990)* 205-215
14. Riehle, D.: Role Model Based Framework Design and Integration. In *Proceedings of the 1998 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'98)*. ACM Press (1998) 117-131
15. Romero, L., Correia, N.: HyperReal: A Hypermedia model for Mixed Reality. *Proceedings of the 14<sup>th</sup> ACM International Conference of Hypertext and Hypermedia (Hypertext 2003)*, ACM Press, 2-9
16. Reenskaug, T: Working with objects. *The OOram Software Engineering Method*. Manning/Prentice Hall 1996.
17. Rossi, G., Nanard, J., Nanard, M: *Engineering Web Applications using Roles*. Technical Report LIRMM, University of Montpellier, May 2004
18. Schwabe, D, Rossi, G.: An object-oriented approach to web-based application design. *Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet*, v. 4#4, October, 1998, 207-225.
19. Schwabe, D, Guimarães, R., Rossi, G.: Cohesive Design of Personalized Web Applications. *IEEE Internet Computing* 6(2): (2002), 34-43
20. Steimann, F.: On the Representation of Roles in Object-Oriented and Conceptual modeling. *Data and Knowledge Engineering* 35 (2000) 83-106
21. The UML home page: [www.omg.org/uml/](http://www.omg.org/uml/)

# Integrating Unnormalised Semi-structured Data Sources

Sasvimol Kittivoravitkul and Peter M<sup>c</sup>Brien

Department of Computing, Imperial College London, London SW7 2AZ

{sk297, pjm}@doc.ic.ac.uk

<http://www.doc.ic.ac.uk/automed>

**Abstract.** Semi-structured data sources, such as XML, HTML or CSV files, present special problems when performing data integration. In addition to the hierarchical structure of the semistructured data, the data integration must deal with the redundancy in semi-structured data, where the same fact may be repeated in a data source, but should map into a single fact in a global integrated schema. We term semi-structured data containing such redundancy as being an unnormalised data source, and we define a normal form for semi-structured data that may be used when defining global schemas. We introduce special functions to relate object identifiers used in the global data model to object identifiers in unnormalised data sources, and demonstrate how to use these functions in query processing, update processing and integration of these data sources.

## 1 Introduction

Areas of application development such as the WWW, electronic commerce, bioinformatics and other scientific disciplines, have led to a growing demand for data representations that support complex, nested and rapidly evolving structures. Often applications in these areas use a **semistructured data (SSD)** data model, such as XML, HTML or one of a variety of flat-file formats (including CSV and TSV). With the proliferation of distributed and heterogeneous SSD, there is a clear need for techniques to perform data integration over these SSD sources, and provide a global unified view of the data.

One of the main tasks in data integration is to define the **mappings** between individual data sources and the unified global view of those sources. Two basic approaches for specifying this mapping are **global-as-view (GAV)** and **local-as-view (LAV)** [10]. The former approach defines the concepts in the global schema as views over the local source schemas whereas the latter approach defines the sources as views over the global schema. Recently, a new approach called **both-as-view (BAV)** [13] has been proposed that specifies a bi-directional mapping between each source and the global schema. Such bi-directional mappings allow data and queries to be translated in either direction from the global schema to the sources, and *vice versa*. This is important, for example, when integrating data in peer-to-peer contexts [14]. The use of BAV has been investigated in the integration of structured data sources [11], and some work has been carried out on integrating XML data sources [12, 20]. The work on the XML integration has



concentrated on specifying schema relationships and made strong assumptions about the data. In particular, they have assumed that there is no redundancy in the data to be integrated.

```

<root>
  <student>
    <name>Ann</name>
    <tutor>Peter</tutor>
    <course code="DB">
      <dept>CS</dept>
      <lecturer>Simon</lecturer>
      <year>1</year>
      <grade>A</grade>
    </course>
    <course code="Stat">
      <dept>MA</dept>
      <lecturer>Jane</lecturer>
      <year>2</year>
      <grade>C</grade>
    </course>
  </student>
  <student>
    <name>Mark</name>
    <tutor>Alex</tutor>
    <course code="DB">
      <dept>CS</dept>
      <lecturer>Simon</lecturer>
      <year>1</year>
      <grade>B</grade>
    </course>
  </student>
</root>

```

(a)  $F_1$  XML file of undergraduates

```

[DB,CS,Ace]
  student=Ann
  level=UG
  year=1
  grade=A
[OS,CS,Ace]
  student=Mark
  level=UG
  year=2
[Stat,MA,Biet]
  student=Ann
  level=UG
  year=2
  grade=C
  student=Mary
  level=PG
  year=4
  grade=A

```

(b)  $F_2$  Text file of students

**Fig. 1.** Example unnormalised semistructured data sources

Fig. 1(a) and 1(b) illustrate two SSD sources which both contain a degree of redundancy, and which overlap with each other. Since they contain redundancy we call them **unnormalised** SSD sources — more precisely we regard anything with less than the SSD equivalent of second normal form as unnormalised. Fig. 1(a) contains an XML file  $F_1$  with details of undergraduate students, where each course a student is sitting is placed within the student element. For each course, there is a record of the department that manages the course, the lecturer of that course, and the year of study and grade that the student has achieved in the course. Note that there is redundancy in this SSD, since

the fact that the CS department runs the DB lectured by Simon is repeated for the two occurrences of that course.

Fig. 1(b) illustrates a structured text file  $F_2$  containing information of courses taken by undergraduate and postgraduate students. The information in Fig. 1(b) is similar to that of the source in Fig. 1(a), but it is structured in different way *i.e.* student information is nested within course information, and provided information about the department building, which is not in  $F_1$ . It avoids the redundancy of  $F_1$ , in that the department of each course is only recorded once, but has its own redundancy in that the level of a student (UG or PG) is repeated for each course a student sits.

When integrating  $F_1$  and  $F_2$  we have to transform at least one of the files, and in particular deal with inverting the hierarchy present in one file to match that of the other. For example, if we choose in our global schema to model courses as containing multiple students (*i.e.* the  $F_2$  view of the data) then when we transform  $F_1$  we would want to have just one course department pair produced for each distinct code and dept pairing that exists in  $F_1$ : *i.e.* produce a set of records containing just two courses, DB and Stat, with two students under the first, and one under the second. Further, for each course, we would only want to have one dept value, but maintain the multiple year and dept associated with each student.

In this paper, we extend the BAV approach to the integration of SSD sources by relaxing assumptions or conditions in the preliminary work *i.e.* allowing unnormalised SSD sources to be integrated. The highlight of our approach is a semistructured data model which includes the notion of key constraints, and a mechanism to deal with redundancy of the data that allows data to be correctly translated in both directions, that is from sources to the global schema and *vice versa*. In contrast, most previous work in semistructured data translation/transformation [3, 5, 15] has focused on retrieving data from data sources to the global schema, and only defined a one way mapping where data can be migrated from source to global schemas, but not vice versa. Popa *et al.* [16, 17] proposed a framework that semi-automatically generates invertible mappings between a relational source and an equivalent nested XML schema. Their mappings are not strictly invertible, since when there is more than one source, the reverse mapping gives back not only an original data but also information acquired by other sources. To our knowledge, no previous work specifically deals with normalisation of SSD sources in the context of data integration. Also, the subject of invertible mappings in SSD integration context has not been fully addressed.

The paper is structured as follows. Section 2 reviews a SSD modelling language called YATTA we shall use for representing SSD data sources. We use the YATTA model as basis for defining normal forms of SSD in Section 3, based on the well known notions of normal forms in relational models. This notion of second normal form is used as a basis for specifying our mappings using the keys in Section 4. It should be noted that it is only the global schema that must obey the second normal form, and the sources may remain unnormalised if we want to take a simple union of data, or they must be in the SSD equivalent of first normal form if actual *data* (as opposed to schema) integration is to take place. Section 5 shows how queries and updates are processed using this mapping approach. Our summary and conclusions are in Section 6.

## 2 The YATTA Data Model

We adopt the **YAT for Transformation-based Approach (YATTA)** model [2] to model SSD sources. YATTA is a variation of the YAT model [4] that has two levels of abstraction, called the **schema level** the **data level**.

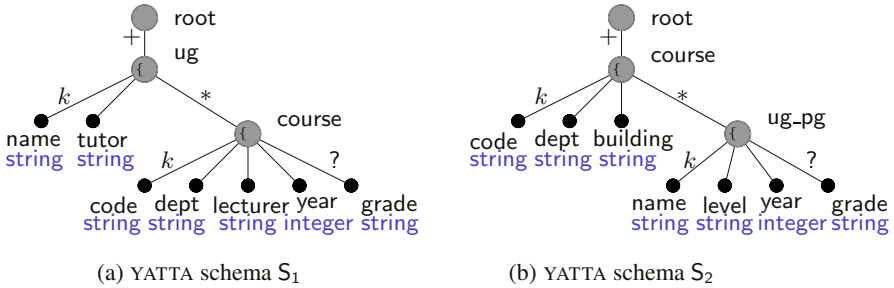


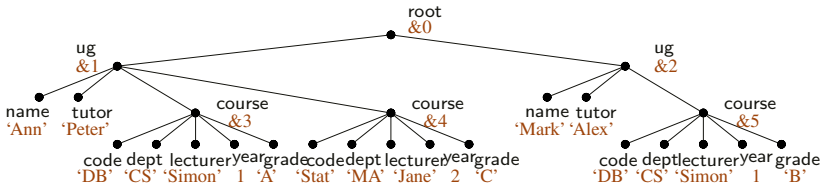
Fig. 2. The YATTA schemas of the XML and text file Fig. 1(a) and 1(b)

A YATTA **schema** represents the structure of a SSD source. Each node in a YATTA schema is labelled with a pair of strings, representing its name and data type. The data type for a leaf node is one of the atomic types which are string, integer and real whereas the data type for a non-leaf node is of type list or set, represented by '[' ]' and '{ }'. Fig. 2(a) and 2(b) represent YATTA schemas for the XML and text files in Fig. 1(a) and 1(b).

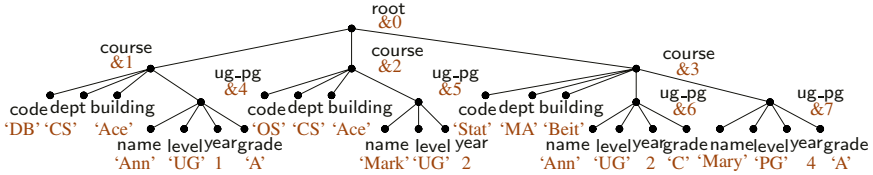
Each YATTA schema edge between nodes  $\langle i, j \rangle$  can be labelled with a cardinality constraint that determines the number of times corresponding nodes  $j$  may occur under each node  $i$  in a YATTA data tree, where '\*' indicates zero or more occurrences, '+' indicates one or more occurrences, '?' indicates zero or one occurrence, and label '1' indicates exactly one occurrence (and is implied if the edge is unlabelled). The symbol 'k' on an edge indicates that a  $j$  is a **key node**, the values of which must be distinct from the values of other  $j$  nodes that appear as siblings of  $i$ . Hence  $k$  also implies the '1' constraint. In  $S_1$ , the key node of the course node is code, and says that each course for a particular ug will have a distinct code value.

Unlike a YATTA schema, a YATTA **data tree** has no labels on its edges. Each node is labelled by a tuple representing its name and value. The values of leaf nodes are the actual data in a data source whereas the values for non-leaf nodes are assigned by the system using **integer identifiers** (denoted by '&' followed by a number e.g. '&0'). The root node is always named root, and its identifier differentiates between particular source files (such as  $D_1$ ) that obey a schema (such as  $S_1$ ).

Fig. 3(a) and 3(b) illustrate YATTA data trees of the XML and text files, and which match the YATTA schemas  $S_1$  and  $S_2$ . For each data tree node there is a corresponding schema node with the same path, such that the data tree node value is compatible with the data type of that schema node. For example, the path  $\langle\langle \text{root}, \text{ug}, \text{course}, \text{grade} \rangle\rangle$  in  $D_1$  leads to three grade data nodes. We will describe the extent of such nodes by listing



(a)  $D_1$  The data tree of the XML file



(b)  $D_2$  The data tree of the text file  $F_2$

Fig. 3. Examples YATTA data trees for Fig. 1

their values along with the identifiers of the parent, and hence  $\langle\langle \text{root}, \text{ug}, \text{course}, \text{grade} \rangle\rangle$  gives  $\{\{\&3, 'A'\}, \{\&4, 'C'\}, \{\&5, 'B'\}\}$ . In the schema, the same path leads to a simple string type node, which matches the type of the second value in each of the tuples in the extent list.

The occurrences of the data node follow the cardinality specified by the symbols on the incoming edges of the corresponding node in the schema. For example, in Fig. 3(a), each student has exactly one tutor, as specified by implied ‘1’ on the incoming edges of tutor in  $S_1$ , and each code of course only exists once for a particular undergraduate student which is also defined name as a key node. The label ‘ $k$ ’ on the incoming edges of name and code in  $S_1$  means the two ug nodes always have different name and no undergraduate student may take two courses with the same code.

### 3 Normal Forms for Semistructured Data Sources

In practice, many SSD sources have the property that each subrecord has a distinguishing attribute or set of attributes that uniquely identifies the subrecord of a given record. We now formalise this idea into the notion of **normal forms** for SSD that may be used when defining global schemas to help avoid data redundancy, inconsistency and undesirable updating anomalies in the integration.

Normal forms have been extensively investigated in the relational model [6] and have recently been extended to SSD [1, 8, 19]. Both [1] and [8] defined a normal form for XML, called XNF, but the two approaches differ. [1] proposed the concept of functional dependency for XML, and defined BCNF for XML documents. In [8], an XML document is in XNF if its specification does not contain potential redundancy w.r.t. a specified set of constraints. Their definition is comparable to the requirement of 3NF

in the relational model. [19] defined a normal form for SSD represented in the XML model, called NS-SS, which appears to be analogous to BCNF. In order to define NS-SS, they introduced the concept of ‘extended functional dependency’, which extends functional dependency in the relation model to support hierarchical data, and the notion of key constraints.

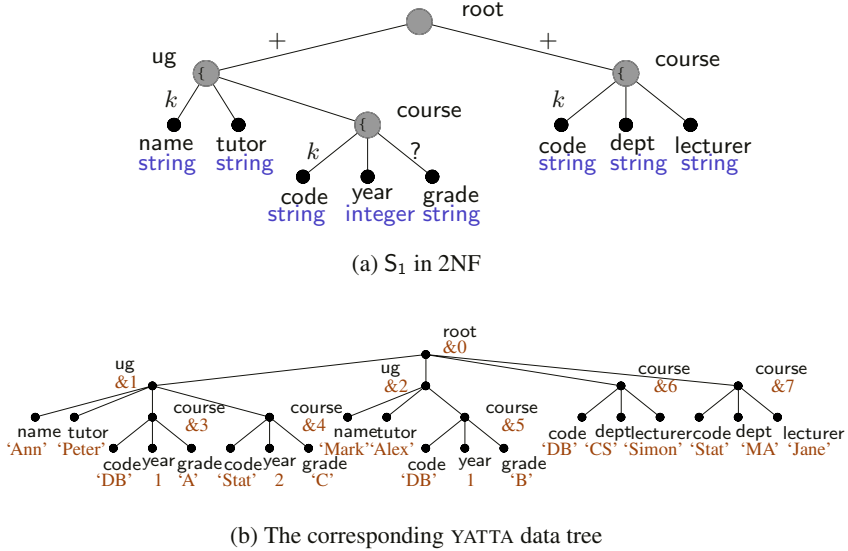


Fig. 4. The 2NF of YATTA source in Fig. 3(a)

The BCNF or 3NF proposed by [1, 8, 19] gives a well-designed data source, but also increase the complexity in accessing a data source, and the use of SSD models is to achieve flexibility *i.e.* not too rigid design. Thus we work with weaker normal forms, comparable to first and second normal forms in the relational model, which are the minimum to achieve data integration.

In the relational model, **first normal form (1NF)** states that each attribute of a relation is functionally determined by its key value, and implies that each relation has a key, and that the non-key attributes take single values. We will define 1NF in SSD as saying that each non-root, non-leaf node in the schema contains at least one key child leaf node (*i.e.* there is at least one *k* beneath each non-leaf node), and that all leaf nodes do not use \* or + cardinality constraints. This means that the non-key leaf nodes can be identified by combining all the key nodes in the path to the non-key leaf node. Schema  $S_1$  obeys our 1NF, and this means we can identify each *ug* and its tutor leaf node by name values, and we can identify each *course* and its dept, lecturer, year and grade leaf nodes, by the combination of its key node values *i.e.* code and name values. Note that in order to truly integrate *data*, as opposed to just the schema holding the data, sources must be in 1NF, since that allows us to identify data values by a **natural key (NK)**

(i.e. a set of values from the real world) rather than the **artificial key (AK)** (i.e. object identifiers) used by the system.

In the relational model, **second normal form (2NF)** states that each attribute of a relation is functionally determined by primary key, but not by any proper subset of the key. In the YATTA model, this corresponds to the schema being in 1NF, with the additional constraint that all of the key nodes are necessary to determine each non-key node.  $S_1$  is not in 2NF, since dept and lecturer are determined by code alone, and not code and name combined.

We can **normalise** schema  $S_1$  to that shown in Fig. 4, by forming a copy of the course under the root, with just those non-key nodes which are determined by code alone being moved to the new course node, the remaining nodes staying as they were in  $S_1$ .

In this paper, we call sources not in the 2NF, **unnormalised** data sources. As in the relational model, the redundancy in unnormalised SSD sources leads to difficulties with updates, and in integration also leads to inefficiencies since the mapping tables based on the keys will contain redundant information. For example, to update the department for the 'DB' course from 'CS' to 'MA' in Fig. 3(a), we are faced with either the problem of searching the tree to find every course containing 'DB' and 'CS' (and changing it) or the possibility of producing an inconsistent result, for example that the department for 'DB' might be given as 'MA' in one record and 'CS' in another. The next section explains how we write mapping rules that take account of this redundancy.

## 4 Mapping Semistructured Data Sources Using Natural Keys

The BAV approach integrates data sources by transforming source schemas into a global schema through sequences of transformations called **pathways**. Each transformation makes a 'delta change' to a schema, adding, deleting, or renaming a single schema node. Each transformation contains a query that specifies the instances of a node in the corresponding data tree. We use the BAV transformation rules for the YATTA model that are defined in [2].

Suppose we want to integrate the 1NF SSD sources in Fig. 1(a) and 1(b). First we design a global schema  $S_g$ , such as that in Fig. 5, and then give mapping rules that define how each node of a global schema can be defined from the source schemas. Based on the approach described in [2], when adding or deleting a node, we describe a scheme for the node which contains the pathway to the node, the type of the node, and the cardinality of the edge that leads to the node from its parent. For example, the scheme of the student node in  $S_g$  is  $\langle\langle\text{root,student,set,+}\rangle\rangle$  and the name is  $\langle\langle\text{root,student,name,string,k}\rangle\rangle$ .

The schemas  $S_1$  and  $S_2$  are transformed into  $S_g$  by applying a pathway of YATTA transformation rules, where each pathway consists of a **growth phase** in which nodes in  $S_g$  that do not exist in the source schema are added, followed by a **shrinking phase** in which nodes that exist in the source schema but not in  $S_g$  are deleted. When a new node is added in the source schema, the query specifies how the instances of a node in the corresponding data tree should be populated. When an existing node is deleted in a schema, the query specifies how the instances of the node in the corresponding data

tree can be restored from the remaining nodes. The mapping of the data sources to the global schema is therefore specified through the queries in a pathway.

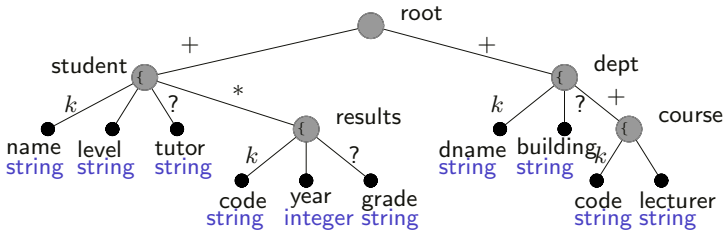


Fig. 5. The global schema  $S_g$

To ensure that source data is correctly mapped with the global schema, the transformations must take into account the following issues:

**Identifier Conflicts** Different data sources might be given different identifiers for the nodes representing the same thing, and the same identifier might be given for different things. For example, the identifier ‘&1’ is assigned to the student node in  $S_1$  in Fig. 3(a) and the course node in  $S_2$  in Fig. 3(b). Hence, in specifying the mapping, a mechanism to resolve these conflicts is required.

**Hierarchical structure** The global schema might be structured in a different way from the source schemas, as shown in the examples. In this case, the mapping therefore involves preserving the relationships between data elements that are implied by the hierarchical structure of the data source.

To resolve identifier conflicts, we apply the concept of a **surrogate keys (SK)** [7], which provides a way of mapping between a NK used in the real world, and an AK used by the system. This mapping is realised using two functions generateGID and generateSID, which are used in the queries of the add and delete transformations, respectively. The functions use the data values (NKs) to associate source identifiers *sids* (AKs) with global identifiers *gids*, which are generated as surrogates.

The generateGID function takes a source schema name, a *sid*, a list of data values and the name of a node that the transformation applied to. The function returns a new OID (*gid*) for every distinct list of data values, and returns the same *gid* for the same list of data values. Hence each *gid* serves as a surrogate for some set of data values. When a global schema is in 2NF, these set of data values are the key values of the node that a transformation applied to.

Conversely, the generateSID takes a source schema name, a list of data values and the name of a node in the global schema. The function returns a source OIDs (*sid*) that have been used in generating *gid* for the same global schema node. The generateSID function can be thought of as the reverse of the generateGID function in the sense that generateGID takes *sid* in the source and generates *gid* for the global view whereas generateSID returns *sid* of the source.

For these functions to be applied in the pathway, the data sources and global schema must be at least in 1NF. This allows the key values in the different sources to be mapped to those of the global schema, thereby allowing data from those sources to be combined. If data sources or global schema are not in 1NF, the integration will just take the union of the respective sources, which is not a real data integration.

To illustrate the generateGID and generateSID functions, we explain how they are used in the pathways  $S_1 \rightarrow S_g$  and  $S_2 \rightarrow S_g$ , an extract from which is shown below. Note that  $S_1$  and  $S_2$  are in 1NF whereas  $S_g$  is in 2NF to avoid redundancy in the integrated data. The student node in  $S_g$  is created by the add transformations ① in  $S_1 \rightarrow S_g$  and ⑤ in  $S_2 \rightarrow S_g$ .

$S_1 \rightarrow S_g$

- ① addYattaNode( $\langle\langle\text{root,student,set,+}\rangle\rangle, [\{r, \text{generateGID}(S_1, u, [n], \text{'student'})\} | \{r, u\} \leftarrow \langle\langle\text{root,ug}\rangle\rangle; \{u, n\} \leftarrow \langle\langle\text{root,ug,name}\rangle\rangle]$ )
- ② addYattaNode( $\langle\langle\text{root,student,name,string,k}\rangle\rangle, [\{\text{generateGID}(S_1, u, [n], \text{'student'})\}, n | \{u, n\} \leftarrow \langle\langle\text{root,ug,name}\rangle\rangle]$ )
- ③ addYattaNode( $\langle\langle\text{root,student,level,string,l}\rangle\rangle, [\{\text{generateGID}(S_1, u, [n], \text{'student'})\}, \text{'ug'} | \{u, n\} \leftarrow \langle\langle\text{root,ug,name}\rangle\rangle]$ )
- ④ addYattaNode( $\langle\langle\text{root,student,tutor,string,?}\rangle\rangle, [\{\text{generateGID}(S_1, u, [n], \text{'student'})\}, t | \{u, n\} \leftarrow \langle\langle\text{root,ug,name}\rangle\rangle; \{u, t\} \leftarrow \langle\langle\text{root,ug,tutor}\rangle\rangle]$ )

$S_2 \rightarrow S_g$

- ⑤ addYattaNode( $\langle\langle\text{root,student,set,+}\rangle\rangle, [\{r, \text{generateGID}(S_2, p, [n], \text{'student'})\} | \{r, c\} \leftarrow \langle\langle\text{root,course}\rangle\rangle; \{c, p\} \leftarrow \langle\langle\text{root,course,ug_pg}\rangle\rangle; \{p, n\} \leftarrow \langle\langle\text{root,course,ug_pg,name}\rangle\rangle]$ )
- ⑥ addYattaNode( $\langle\langle\text{root,student,name,string,k}\rangle\rangle, [\{\text{generateGID}(S_2, p, [n], \text{'student'})\}, n | \{c, p\} \leftarrow \langle\langle\text{root,course,ug_pg}\rangle\rangle; \{p, n\} \leftarrow \langle\langle\text{root,course,ug_pg,name}\rangle\rangle]$ )

The IQL [18] query in ① finds in the generator  $\{r,u\} \leftarrow \langle\langle\text{root,ug}\rangle\rangle$  the tuples  $\{\&0, \&1\}$ ,  $\{\&0, \&2\}$ , and  $\{u,n\} \leftarrow \langle\langle\text{root,ug,name}\rangle\rangle$  the tuples  $\{\&1, \text{'Ann'}\}$ ,  $\{\&2, \text{'Mark'}\}$ . Then the generateGID function is called with  $(S_1, \&1, [\text{Ann}], \text{'student'})$  and  $(S_1, \&2, [\text{Mark}], \text{'student'})$ . The function generates &101 and &102 as new global integer identifiers (*gids*) for each list of the data value, [*Ann*] and [*Mark*], which are the key values of student in  $S_g$ . Hence the list  $[\{\&0, \&101\}, \{\&0, \&102\}]$  will be associated with  $\langle\langle\text{root,student}\rangle\rangle$ . A similar analysis for transformation ② will give list  $[\{\&101, \text{'Ann'}\}, \{\&102, \text{'Mark'}\}]$  being associated with  $\langle\langle\text{root,student,name}\rangle\rangle$ , and so on for the remaining transformations. The mapping of *sids* to *gids* for the student node through the values of the name node, which is the key node of student, is shown in Fig. 6 (though at this stage, the  $S_2$  part of the graph should be ignored).

Similarly, the query in transformation ⑤ finds the tuples:  $\{\&0, \&1\}$ ,  $\{\&0, \&2\}$ ,  $\{\&0, \&3\}$  from  $\{r,c\} \leftarrow \langle\langle\text{root,course}\rangle\rangle$ , then the tuples  $\{\&1, \&4\}$ ,  $\{\&2, \&5\}$ ,  $\{\&3, \&6\}$ ,  $\{\&3, \&7\}$  from  $\{c,p\} \leftarrow \langle\langle\text{root,course,ug_pg}\rangle\rangle$ , and finally the tuples  $\{\&4, \text{'Ann'}\}$ ,  $\{\&5, \text{'Mark'}\}$ ,  $\{\&6, \text{'Ann'}\}$ ,  $\{\&7, \text{'Mary'}\}$  from  $\{p,n\} \leftarrow \langle\langle\text{root,course,ug_pg,name}\rangle\rangle$ . This causes generateGID to receive  $(S_2, \&4, [\text{Ann}], \text{'student'})$ ,  $(S_2, \&5, [\text{Mark}], \text{'student'})$ ,  $(S_2, \&6, [\text{Ann}], \text{'student'})$  and  $(S_2, \&7, [\text{Mary}], \text{'student'})$ . Since the *gids* for [*Ann*] and [*Mark*] already exist, the function returns &101 and &102 and generates a new *gid*



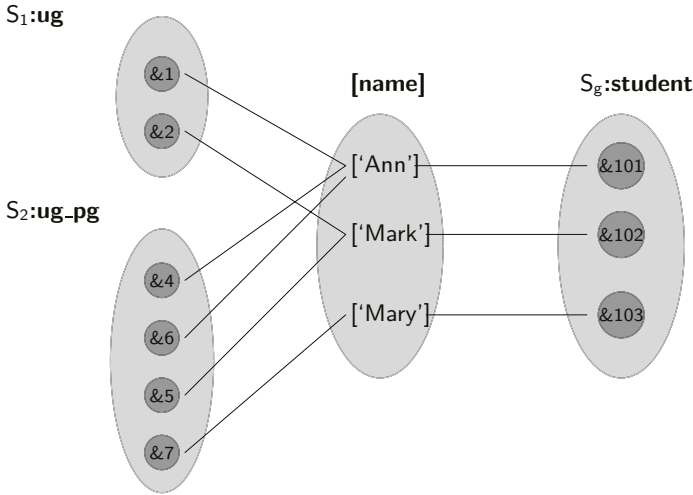


Fig. 6. The mapping between *sids* and *gids* of the student node

&103 for [Mary]. Transformation ⑤ gives a list  $\{ \{&0, &101\}, \{&0, &102\}, \{&0, &103\} \}$  being associated with the scheme  $\langle \langle \text{root}, \text{student} \rangle \rangle$ .

As illustrated in Fig. 6, the generateGID function groups together the *ug* and *ug.pg* nodes related to the same name, and creates a *gid* for each group. This resolves the identifier conflicts among data sources and minimises the data redundancy in the integration. The data that is related to name such as tutor, is also put under the student node related to such name as specified in the query in the transformation ④. The *gids* created by the generateGID function as well as its parameters can be stored as shown in Fig. 7 and 8.

Below are the transformations in the shrinking phase of  $S_1 \rightarrow S_g$  that remove the tutor, name and *ug* nodes in  $S_1$  (we omit details of how before these transformations, grade, year, lecturer, dept, code and course are deleted). The generateSID function is applied in this phase to allow the **reverse** transformation to recover the original data. This reversibility ensures *information preservation* in the transformation. Importantly,

local schema	sid	NK	name
$S_1$	[ &1 ]	[ 'Ann' ]	student
$S_1$	[ &2 ]	[ 'Mark' ]	student
$S_2$	[ &4 ]	[ 'Ann' ]	student
$S_2$	[ &5 ]	[ 'Mark' ]	student
$S_2$	[ &6 ]	[ 'Ann' ]	student
$S_2$	[ &7 ]	[ 'Mary' ]	student
.	.	.	.

Fig. 7. *sids* and the data values

gid	NK	name
[ &101 ]	[ 'Ann' ]	student
[ &102 ]	[ 'Mark' ]	student
[ &103 ]	[ 'Mary' ]	student
.	.	.
.	.	.
.	.	.
.	.	.

Fig. 8. *gids* and the data values

it allows data, queries and updates to be automatically migrated or translated in either direction between source and the global schemas.

$S_1 \rightarrow S_g$

- ⑦ delYattaNode( $\langle\langle\text{root,ug,tutor,string,1}\rangle\rangle, [\{\text{generateSID}(S_1, [n], \text{'student'}), t\} \mid \{s, n\} \leftarrow \langle\langle\text{root, student, name}\rangle\rangle; \{s, t\} \leftarrow \langle\langle\text{root, student, tutor}\rangle\rangle; \{s, \text{'ug'}\} \leftarrow \langle\langle\text{root, student, level}\rangle\rangle]$ )
- ⑧ delYattaNode( $\langle\langle\text{root,ug,name,string, } k\rangle\rangle, [\{\text{generateSID}(S_1, [n], \text{'student'}), n\} \mid \{s, n\} \leftarrow \langle\langle\text{root, student, name}\rangle\rangle; \{s, \text{'ug'}\} \leftarrow \langle\langle\text{root, student, level}\rangle\rangle]$ )
- ⑨ delYattaNode( $\langle\langle\text{root,ug,set,+}\rangle\rangle, [\{r, \text{generateSID}(S_1, [n], \text{'student'})\} \mid \{r, s\} \leftarrow \langle\langle\text{root, student}\rangle\rangle; \{s, n\} \leftarrow \langle\langle\text{root, student, name}\rangle\rangle; \{s, \text{'ug'}\} \leftarrow \langle\langle\text{root, student, level}\rangle\rangle]$ )

Transformation ⑦ removes the tutor node from ug. The query in the transformation states that the values of tutor in  $S_1$  can be restored from tutor in  $S_g$ . It finds the tuples  $\{\&101, \text{'Ann'}\}$ ,  $\{\&102, \text{'Mark'}\}$ ,  $\{\&103, \text{'Mary'}\}$  from  $\{s,n\} \leftarrow \langle\langle\text{root, student, name}\rangle\rangle$ , then the tuples  $\{\&101, \text{'Peter'}\}$ ,  $\{\&102, \text{'Alex'}\}$  from  $\{s,t\} \leftarrow \langle\langle\text{root, student, tutor}\rangle\rangle$ , then the tuples  $\{\&101, \text{'ug'}\}$ ,  $\{\&102, \text{'ug'}\}$  from  $\{s,\text{'ug'}\} \leftarrow \langle\langle\text{root, student, level}\rangle\rangle$ . The generateSID function is called with  $(S_1, [\text{'Ann'}], \text{'student'})$  and  $(S_1, [\text{'Mark'}], \text{'student'})$ . The function then looks up the key values in the table of Fig. 7, and restores the values of the ug node in  $S_1$  with &1 and &2, which are the *sids* related to 'Ann' and 'Mark' in the source  $S_1$ . The queries in ⑧ and ⑨ can be read in a similar manner.

## 5 Queries and Updates over the Mapping

After defining the mappings of data sources to the global schema, one may want to query or update data sources through the global schema. The bi-directional mappings allow queries and updates posed on the global schema to be automatically translated to ones poses on data sources.

### 5.1 Query Translation

To translate a query  $Q_g$  posed on the global schema to a query on data source  $S_x$ , we need only consider delete transformations in the pathway  $S_g \rightarrow S_x$ . Every deleted construct appearing in the query  $Q_g$  are substituted by the query in the transformations. For example, suppose we pose the query  $Q_1$  on  $S_g$  asking for all students, which in the IQL would take the form:

$$Q_1 = [\{x, y\} \mid \{x, y\} \leftarrow \langle\langle\text{root, student, name}\rangle\rangle]$$

The pathways  $S_g \rightarrow S_1$  and  $S_g \rightarrow S_2$  can be automatically derived from  $S_1 \rightarrow S_g$  and  $S_2 \rightarrow S_g$ , respectively, by replacing delete for add, and replacing add for delete. Below are the inverse steps ②–① in transformations ①–④, and ⑥–⑤ in transformations ⑤–⑥.

$S_g \rightarrow S_1$

- ④ delYattaNode( $\langle\langle\text{root,student,tutor,string,?}\rangle\rangle, [\{\text{generateGID}(S_1, u, [n], \text{'student'}), t\} \mid \{u, n\} \leftarrow \langle\langle\text{root, ug, name}\rangle\rangle; \{u, t\} \leftarrow \langle\langle\text{root, ug, tutor}\rangle\rangle]$ )

- ③  $\text{delYattaNode}(\langle\langle\text{root,student,level,string,1}\rangle\rangle, [\{\text{generateGID}(S_1, u, [n], \text{'student'}), \text{'ug'}\} | \{u, n\} \leftarrow \langle\langle\text{root, ug, name}\rangle\rangle])$
- ②  $\text{delYattaNode}(\langle\langle\text{root,student,name,string, } k\rangle\rangle, [\{\text{generateGID}(S_1, u, [n], \text{'student'}), n\} | \{u, n\} \leftarrow \langle\langle\text{root, ug, name}\rangle\rangle])$
- ①  $\text{delYattaNode}(\langle\langle\text{root,student,set,+}\rangle\rangle, [\{r, \text{generateGID}(\text{'student'}, u, [n])\} | \{r, u\} \leftarrow \langle\langle\text{root, ug}\rangle\rangle; \{u, n\} \leftarrow \langle\langle\text{root, ug, name}\rangle\rangle])$

$S_g \rightarrow S_2$

- ⑥  $\text{delYattaNode}(\langle\langle\text{root,student,name,string, } k\rangle\rangle, [\{\text{generateGID}(S_2, p, [n], \text{'student'}), n\} | \{c, p\} \leftarrow \langle\langle\text{root, course, ug\_pg}\rangle\rangle; \{p, n\} \leftarrow \langle\langle\text{root, course, ug\_pg, name}\rangle\rangle])$
- ⑤  $\text{delYattaNode}(\langle\langle\text{root,student,set,+}\rangle\rangle, [\{r, \text{generateGID}(S_2, p, [n], \text{'student'})\} | \{r, c\} \leftarrow \langle\langle\text{root, course}\rangle\rangle; \{c, p\} \leftarrow \langle\langle\text{root, course, ug\_pg}\rangle\rangle; \{p, n\} \leftarrow \langle\langle\text{root, course, ug\_pg, name}\rangle\rangle])$

To translate the query  $Q_1$  into source ones, the construct  $\langle\langle\text{root, student, name}\rangle\rangle$  is replaced by the queries  $q_1$  and  $q_2$  in transformations ② in the pathway  $S_1 \rightarrow S_g$  and ⑥ in the pathway  $S_2 \rightarrow S_g$ , combined by the OR operator [9].

$$Q_1 = [\{x, y\} | \{x, y\} \leftarrow (q_1 \text{ OR } q_2)]$$

The query  $q_1$  returns a list  $[\{\&101, \text{'Ann'}\}, \{\&102, \text{'Mark'}\}]$  whereas the query  $q_2$  gives a list  $[\{\&101, \text{'Ann'}\}, \{\&102, \text{'Mark'}\}, \{\&103, \text{'Mary'}\}]$  as described in the previous section. Using union semantics for the OR operator, we therefore get are all students from both data sources:

$$Q_1 = [\{\&101, \text{'Ann'}\}, \{\&102, \text{'Mark'}\}, \{\&103, \text{'Mary'}\}]$$

In this simple example, the `generateGID` function reduces the redundancy by ensuring that 'Ann' is returned only once. Without the `generateGID` function, the name 'Ann' would appear three times as a result of different identifiers of student nodes in the data sources.

In general, the `generateGID` function combines data that requires merging but has different identifiers in the sources (*e.g.* student nodes with identifiers &2 and &5, which are related to 'Mark'), and avoids combining distinct data that has the same identifier in different sources (*e.g.* course nodes with identifiers &2 in  $D_1$  and  $D_2$ , which are related to 'Mark' and 'OS', respectively). In addition, the function allows related information in different sources to be brought together by the key values. For example, the information about the department building and the lecturer are in different sources, but by posing the query  $Q_2$  below on  $S_g$ ,

$$Q_2 = [\{x, y, z, w\} | \{x, y\} \leftarrow \langle\langle\text{root, dept, building}\rangle\rangle; \{x, c\} \leftarrow \langle\langle\text{root, dept, course}\rangle\rangle; \{c, z\} \leftarrow \langle\langle\text{root, dept, course, code}\rangle\rangle; \{c, w\} \leftarrow \langle\langle\text{root, dept, course, lecturer}\rangle\rangle]$$

the query results in a list of the departments, the building names, the course codes, and the lectures for the courses as shown below, since they are joined by the natural keys identifying dept and course.

$$Q_2 = [\{\text{'CS'}, \text{'ACE'}, \text{'DB'}, \text{'Simon'}\}, \{\text{'CS'}, \text{'ACE'}, \text{'OS'}, \text{'Void'}\}, \{\text{'MA'}, \text{'Beit'}, \text{'Stat'}, \text{'Jane'}\}]$$

## 5.2 Update Translation

Applying an update  $U$  requires giving the scheme of the construct to be updated, and the new value of the construct tuple. For example, to change the tutor which is associated to student with identifier &101 to 'Fred', the update statement can be written as:

$$U_1 = \text{update}(\langle\langle\text{root, student, tutor}\rangle\rangle, \{\&101, \text{'Fred'}\})$$

Translating updates posed on a global schema to ones on data sources is different from translating queries described earlier. In update translation, all delete rules in pathways  $S_x \rightarrow S_g$  that use the scheme of the construct to be updated should be retrieved. For the example, only (7) in the pathway  $S_1 \rightarrow S_g$  matches the criteria for  $U_1$ , since its IQL query contains  $\langle\langle\text{root, student, tutor}\rangle\rangle$ . The data source is then updated by processing the queries accompanying the delete transformation using the new value of the construct tuple. In the example, the query accompanying (7) in the pathway  $S_1 \rightarrow S_g$  is processed using the new value  $\{\&101, \text{'Fred'}\}$ . The association between the *gid* &101 and *sid* &1 is obtained by the generateSID function. The value  $\{\&1, \text{'Peter'}\}$  in the  $\langle\langle\text{root, ug, tutor}\rangle\rangle$  of  $S_1$  is then updated to  $\{\&1, \text{'Fred'}\}$ .

Note that there may be multiple source nodes to update for some update statements. For example, with the update on  $S_g$  to change the department name of the course with value &110 to 'Math' such as the following:

$$U_2 = \text{update}(\langle\langle\text{root, course, dept}\rangle\rangle, \{\&110, \text{'Math'}\})$$

There would be potentially several data source nodes that would be returned when the rules contain  $\langle\langle\text{root, course, dept}\rangle\rangle$  are found.

## 6 Summary and Conclusions

In this paper we have considered the issue of normalisation of SSD as part of a data integration process. We defined a SSD first normal form that allows us to identify data values in sources by the use of a natural key, and hence perform data (as opposed to just schema) integration. It is proposed that the global schema be in second normal form to allow updates to data sources to be made in a consistent manner, and to minimise the size of the mapping tables between source and global schema object identifiers. The normal forms considered in this paper are weaker (and simpler) than those defined in the literature, but if required, our approach could easily be extended to use higher normal forms, such as those defined in [19]. However, 2NF is sufficient for our approach to work, and provided a data source is not denormalised by the data integration rules (*i.e.* brought down from a higher normal form to 2NF), then no update anomalies will be introduced by the process of data integration.

We introduced the functions generateGID and generateSID, which use a natural key to relate object identifiers in the 1NF (or higher) source schemas to identifiers in the global schema, and showed how these functions are used in the bi-directional transformation pathways of the BAV approach to data integration. The generateGID function allows data from different sources that require merging/joining to be correctly combined. It solves the identifier conflicts among local data sources, mapping them to a single global schema identifier, by relating them via a natural key. This reduces

redundancy in the integrated schema, and allows updates to be performed without introducing anomalies. The `generateSID` function allows the original data sources to be restored. It ensures the pathways are reversible, and therefore allows updating from the global schema to the sources, and allows data and queries to be migrated in both directions — a functionality particularly important in peer-to-peer contexts.

Our approach has been implemented in the AutoMed data integration system (details of which may be found at (<http://www.doc.ic.ac.uk/automed>). The `generateGID` and `generateSID` functions have been integrated into the query processor to allow the system to correctly combine data from different sources, to support updating of data sources and to enable the original data sources to be restored. For the future work, we will be building larger case studies focusing on the integration of biological data sources, which are often held in the form of flat files, HTML or XML.

## References

1. M. Arenas and L. Libkin. A normal form for xml documents. *ACM Transactions on Database Systems*, 29(1):195–232, 2004.
2. M. Boyd, S. Kittivoravitkul, C. Lazanitis, P.J. McBrien, and N. Rizopoulos. AutoMed: A BAV data integration system for heterogeneous data sources. In *Proc. CAiSE2004*, volume 3084 of *LNCS*, pages 82–97. Springer-Verlag, 2004.
3. V. Christophides, S. Cluet, and J. Siméon. On wrapping query languages and efficient xml integration. *SIGMOD Rec.*, 29(2):141–152, 2000.
4. S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! In *Proc. SIGMOD'98*, pages 177–188. ACM Press, 1998.
5. S. Cluet and J. Siméon. Data integration based on data conversion and restructuring. Technical report, Verso database group- INRIA, France, 1997.
6. C.J. Date. *An Introduction to Database Systems*. Addison-Wesley, 8th edition edition, 2004.
7. C.J. Date, H. Darwen, and D. McGoveran. *Relational Database: Selected Writings 1994–1997*. Addison-Wesley, 1998.
8. D.W. Embley and W.Y. Mok. Developing xml documents with guaranteed “good” properties. In *Proc. 20th ER*, pages 426–441, 2001.
9. E. Jasper, N. Tong, P.J. McBrien, and A. Poulouvasilis. View generation and optimisation in the AutoMed data integration framework. In *Proc. Baltic DB&IS04*, volume 672 of *Scientific Papers*, pages 13–30. Univ. Latvia, 2004.
10. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS'02*, pages 233–246. ACM, 2002.
11. P.J. McBrien and A. Poulouvasilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99*, volume 1626 of *LNCS*, pages 333–348. Springer, 1999.
12. P.J. McBrien and A. Poulouvasilis. A semantic approach to integrating XML and structured data sources. In *Proc. CAiSE'01*, volume 2068 of *LNCS*, pages 330–345. Springer, 2001.
13. P.J. McBrien and A. Poulouvasilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, pages 227–238. IEEE, 2003.
14. P.J. McBrien and A. Poulouvasilis. Defining peer-to-peer data integration using both as view rules. In *Proc. DBISP2P, at VLDB'03*, Berlin, Germany, 2003.
15. Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 413–424. Morgan Kaufmann Publishers Inc., 1996.
16. L. Popa, Y. Velegrakis, R.J. Miller, M.A. Hernandez, and R. Fagin. Translating web data. In *Proc 28th VLDB*, pages 598–609, 2002.

17. L. Popa, Y. Velegrakis, R.J. Miller, M.A. Hernandez, and R. Fagin. Translating web data. Technical report, Department of Computer Science, University of Toronto, 2002.
18. A. Poulouvassilis. The automed intermediate query language. Technical report, Department of Computer Science, Birkbeck College, 2001.
19. X. Wu, T.W. Ling, S.Y. Lee, M. Lee, and G. Dobbie. Nf-ss: A normal form for semistructured schema. In *ER 2001 Workshops*, pages 292–305, 2001.
20. L. Zamboulis and A. Poulouvassilis. Using automed for xml data transformation and integration. In Z. Bellahsene and P.J. McBrien, editors, *Proc. DIWeb04, CAiSE Workshop Proceedings Volume 3*, pages 58–69, 2004.

# Model Transformations in the Development of Data-Intensive Web Applications

Davide Di Ruscio and Alfonso Pierantonio

Dipartimento di Informatica,  
Università degli Studi di L'Aquila,  
67100 L'Aquila, Italy  
{diruscio, alfonso}@di.univaq.it

**Abstract.** Over the last few years, Web-based systems became commonplace. Despite the complexity and the economic significance of such applications, current practice does not always apply robust and well-understood principles. Model driven architecture (MDA) separates the application logic from the underlying platform technology and represents them with precise semantic models. Web application development therefore has potentially the most to gain from adopting such techniques that can offer a greater return on development time and quality factors than traditional approaches. In particular, the paper presents model-driven transformations between platform-independent (conceptual descriptions of Web applications) and platform-specific (Model-View-Controller conformant) models. The design of such transformations is documented (and possibly animated) through mathematically rigorous specifications given by means of Abstract State Machines.

## 1 Introduction

Over the last few years, Web-based systems became commonplace and underwent frequent modifications due to technological and commercial urges. Web sites rapidly evolved from simple collections of static pages to data-intensive applications which rely on dynamic contents usually stored in databases enabling a much wider range of interaction.

Despite the complexity and the economic significance of such applications, current practice does not always apply robust and well-understood principles. Model driven architecture [15] (MDA) separates the application logic from the underlying platform technology and represents them with precise semantic models. Web application development therefore has potentially the most to gain from adopting such techniques that can offer a greater return on development time and quality factors than traditional approaches.

In this paper, we describe a systematic approach to model-driven development of data-intensive Web applications meant as hybrid between hypermedia and information systems [13]. Starting from a suitable UML profile, called

Webile [24], conceptual descriptions of these systems are given as Platform Independent Models (PIMs), i.e. abstract descriptions that do not refer to the technologies they assume to exist. The process of transforming a PIM to obtain concrete implementations on the target architecture described by Platform Specific Models (PSMs) is the ultimate consequence of shifting the focus of software development from coding to modeling. Different PSMs can be generated from a Webile model in order to describe different aspects of J2EE Web applications designed according to the Model-View-Controller [14] architectural pattern.

Model transformation presents intrinsic difficulties. It requires the ability to simulate arbitrary algorithms on their natural levels of abstraction, without implementing them, makes Abstract State Machines [7] (ASMs) appropriate for high-level system design and analysis [5] and a candidate for specifying model transformation as well. Generating models in a formal setting can facilitate information traceability, reuse and evolution of software systems, but also represents a basis to reason about the intuitions encoded into unambiguous transformation descriptions. Furthermore, the ASM execution environment [2] represents an open framework with built-in support to syntax-tree manipulation (see [3]) useful for both tool integration and automatic transformation.

The paper is organized as follows. The next section illustrates an extended version of the Webile profile, which is used for the description of PIMs. Section 3 presents the founding elements for modeling J2EE Web applications designed according to the MVC architectural pattern. After introducing some basics about the ASMs, next section presents the ASM rules for transforming Webile models. Sect. 6 relates the work presented in this paper with other approaches. Finally, the last section draws some conclusions.

## 2 Webile

Webile [24] is a UML profile for describing in a uniform and conceptual way the proper aspects of data-intensive Web applications without referring to platform-specific assets. Leveraging the recurrency of certain application patterns which typically compose Web applications permits to raise the level of abstraction adopting a model-driven development whose main artifacts are models. These models are supposed to span the entire life cycle of a software system and ease the software production and maintenance tasks.

Descriptions encompass several concerns by capturing data, pages and navigation into extended class diagrams. In particular, data are given similarly to E/R models exploiting stereotyped classes and associations to model entities and relations, respectively. The profile prescribes the `«DataEntity»`, `«DataRelation»`, `«DataStrongRelation»` and `«DataAttribute»` stereotypes for modeling data. For instance, in Fig. 1 the elements contained in the dotted area, represent a



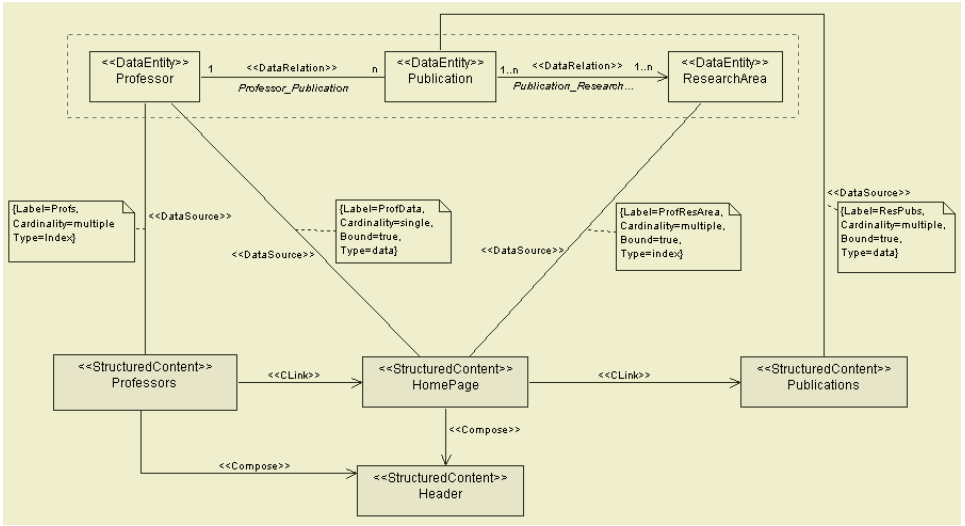


Fig. 1. A fragment of an academic site

simplified<sup>1</sup> conceptual data model of an academic site fragment, where professors (`Professor`) can have different publications (`Publication`), each belonging to one or more research areas (`ResearchArea`).

Pages and their fragments are denoted by means of `StructuredContent` stereotyped classes that are eventually associated with data entities providing contents by means of `DataSource` stereotyped associations. These associations are qualified with a collection of tagged values, amongst them `Cardinality` describes the cardinality of the items to be included in the content, i.e. whether the content consists of a single item or a list of them. In the figure, the `Professors` structured content contains the list of all professors in the database, which are retrieved through the associated entity `Professor`, in contrast with `HomePage` which contains information about one professor, respectively, because of the different specified cardinalities. Relevant aspects of the data source association affect the way the data are retrieved to form structured contents. In fact, different data source associations converging on the same structured content and denoted by the same tagged value `Label` define the same query operation (see Sect. 5). On the contrary, in `HomePage` two different query operations are defined, because the labels on the associations with `Professor` and `ResearchArea` are different.

Hyperlinks are modeled by means of the `CLink` and `NCLink` stereotyped associations which denote contextual and non-contextual links, respectively. The main difference among them lies in the fact that the formers propagate parameters from the source structured content to the target one. These

<sup>1</sup> For presentational purposes, we omitted attributes and other information which are not relevant at this stage of the discussion.

parameters are used when data source associations have the tagged value `Bound` set to `..` to filter the data retrieved from the corresponding entities. For instance, in Fig. 1 the contextual link going out from `Professors` allows the user to select a single professor in order to access her/his personal profile in `HomePage`, which is collected by means of the `<<DataSource>>` stereotyped associations with the entities `Professor` and `ResearchArea`. Analogously, the contextual link outgoing from `HomePage` provides with the access to `Publications` of the selected research area. Non contextual links are much simpler since they connect structured contents which are not semantically correlated.

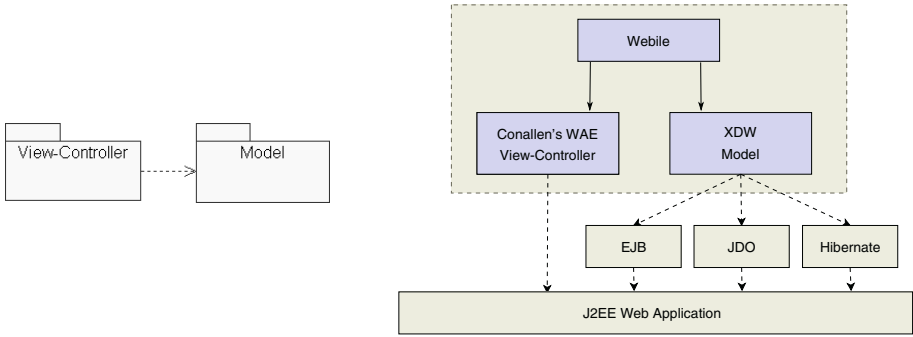
Support to modularity is also important to achieve pragmatic qualities. In fact, the structured contents do not describe only pages but also portion of them, as with `Header` which is shared by the `Professors` and `HomePage` structured contents via the correspondent `<<compose>>` stereotyped associations. Further stereotypes have been defined to cover additional aspects. In particular, the association `<<DataEntry>>` between a page and some data entities is used to declare data entry forms. Moreover, structured contents can be subject to authentication/authorization in order to secure certain contents accessible only to specific users and/or groups specified by the `<<User>>` and/or `<<Group>>` stereotyped classes. Due to space limitations, we did not report data entry forms and protected pages in the examples. Finally, a relevant case study has been produced by modeling a large institutional site in order to validate the expressiveness of the profile (see [12]).

The Webile profile was originally devised to generate code directly from models in an `..` fashion without any human intervention. The approach has shown immediately problems not limited to poor consistency and traceability between models and code, as the formers start to diverge from the latter as soon as changes are operated on the generated system. Thus, the approach has been considerably extended introducing proper model transformations able to map Webile models into model chains which, at different level of abstractions, are descriptions of the chosen implementation.

### 3 Describing PSMs

MVC is an architectural pattern which aims at minimizing the degree of coupling between elements to relate the user interface to underlying data models in an effective way. Increasingly, the MVC pattern is used in program development with object-oriented languages and in organizing the design of J2EE Web applications proposing a three-way factoring paradigm based on the following

- the model holds all data relevant to domain entity or process, and performs behavioral processing on that data;
- the view displays data contained in the model and maintains consistency in the presentation when the model changes; and
- the controller is the glue between view and model reacting to significant events in the view, which may result in manipulation of the model.



**Fig. 2.** Different Views of the MVC pattern

The description of PSMs referring to the J2EE platform may distinguish the model from the view and the controller. This separation of concerns is motivated by the abundance of persistence frameworks, such as EJB [11] and JDO [18] to mention a few, which suggests further refinements of the model into more specific PSMs retaining the view-controller design (see Fig. 2). According to the figure, a Webile specification is mapped into platform-specific descriptions of the view-controller and the model, respectively. This mapping is automatic and mathematically defined by executable ASM transition rules as described in Sect. 5. In the proposed approach, the View-Controller package (see Fig. 2) is given by means of Conallen’s Web Applications Extension [10] (WAE) whereas the Model package is given by means of the data part of Webile opportunely extended to some abstraction for realizing given business tier patterns [1].

**3.1 View-Controller: Conallen’s WAE**

The Web Application Extension (WAE) is an extension of UML for modeling Web applications proposed by J.Conallen. Web pages are modeled by giving both server-side and client-side aspects by means of `<<Server Page>>` and `<<Client Page>>` stereotyped classes, respectively. A server page can be associated with other server-side objects, i.e. database, middle-tier components and so on, although we are not going to model data aspects here. The `<<Client Page>>` stereotype represents a HTML page which is usually associated with other client or server pages. In the last case the `<<build>>` stereotyped association is used to state that a server page builds a client one. An hyperlink between pages is modeled by a `<<link>>` stereotyped association. If the hyperlink includes parameters, they are modelled as link attributes of the association. A directional relationship between one server page and another server or client page is modeled by the `<<forward>>` stereotyped association. This association represents the delegation of processing client’s requests for a resource to another server-side page and it is a pivotal aspect proper of the view-controller metaphor.

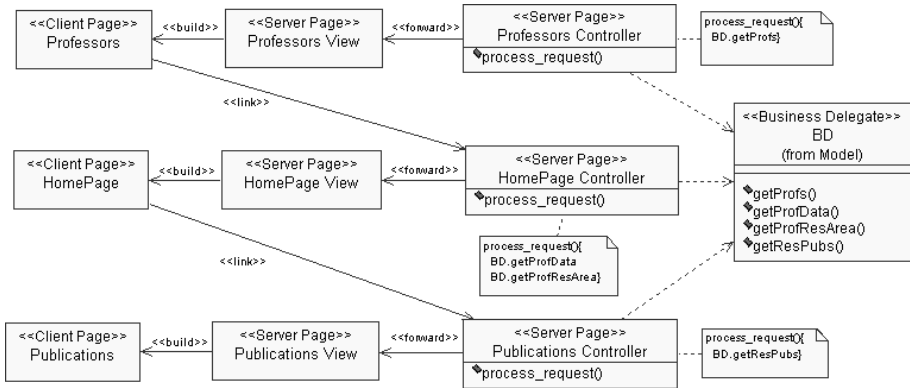


Fig. 3. Conallen’s View-Controller description

In fact, referring to Fig. 3 and according to the adopted pattern, client requests are processed by the controller server pages which perform the data retrieval by invoking the proper operations on the business delegate object (as explained in the next section). Each controller declares exactly the operation which must be invoked according to the data source associations in the conceptual model, e.g. the server page class Home Page Controller depends on the methods getProfData() and getProfResArea() to retrieve the data. Once the data are available to the controller, the request is forwarded to the corresponding view server page. In particular, the figure illustrates how to implement the application logic of the system described in Fig. 1 by means of several views and controllers; each structured content is mapped to a pattern consisting of linked client page, view and controller server pages. Alternatively, the front controller pattern [1], i.e. a unique controller which serves as a centralized access point for requests and link, could have been adopted. It is a solution which is widely used by software developers, which encodes information about the navigation in the url requests, thus is less convenient to illustrate how the navigation in Webile is propagated during model transformation.

Finally, the idea of adopting Conallen’s approach for specifying PSMs is not novel, since it mainly represents the implementation and is therefore suitable for PSMs rather than PIMs [21, 22].

### 3.2 Model: eXtended Data Webile

This section presents how to describe the Model component of the MVC pattern by means of an extension of the data part of Webile, called eXtended Data Webile (XDW). A better maintenance and flexibility in accessing business services requires specific abstraction layers as the ones realized by means of the business delegate and the transfer object design patterns [1]. In particular, the business delegate hides implementation details of the business service and encapsulates access and lookup mechanisms; whereas the transfer object serves to optimize data

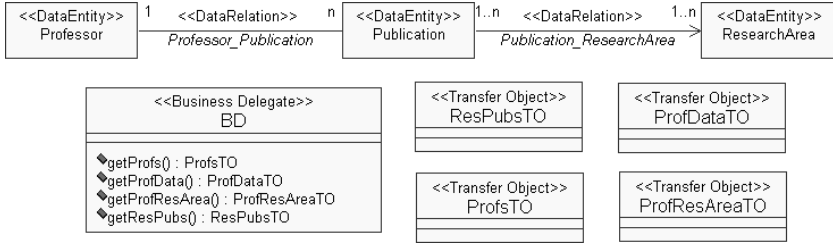


Fig. 4. XDW Model description

transfer across tiers. Instead of sending or receiving individual data elements, a transfer object contains all the data elements in a single structure required by the request or response. To summarize, a controller can access business services by performing requests to a business delegate which implements the services and returns the result as a transfer object. For instance, Fig. 4 depicts a diagram which describes by means of XDW the Model components of the application which has been modeled in Fig. 1. It comprehends only the data aspects of the original model and additionally introduces the business delegate and a transfer object for each different query operation defined within the business delegate. To understand how such elements are defined, let us consider the data source association in Fig. 1 labeled *ProfData* between *HomePage* and *Professor*, this association defines the query operation in the business delegate called *getProfData()* which returns a transfer object of type *ProfDataTO*. In order to keep a certain degree of abstraction, the query operations in the business delegate are specified by means of relational algebra expressions which are computed by ASMs rules presented and commented in Sect. 5.

### 4 Abstract State Machines

Due to space limitation, we only briefly introduce ASMs here insisting on few introductory aspects. For more information, the reader is referred to [6, 7]. ASMs bridge the gap between specification and computation by providing more versatile Turing-complete machines. The ability to simulate arbitrary algorithms on their natural levels of abstraction, without implementing them, makes ASMs appropriate for high-level system design and analysis (see [5]). Additionally, ASMs are executable and several compilers and tools are available both from academy and industry.

ASMs form a variant of first-order logic with equality, where the fundamental concept is that functions are defined over a set  $\mathcal{U}$  and can be changed point-wise. The set  $\mathcal{U}$  referred to as the *universe* in ASM terminology, always contains the distinct elements  $\dots, \dots, \dots$ , and  $\dots$ . Apart from these,  $\mathcal{U}$  can contain numbers, strings, and possibly anything, depending on the application domain.

Being slightly more formal, we define the *lambda* of a system as a mapping from a signature  $\Sigma$  (which is a collection of function symbols) to actual functions.

We write  $f_\lambda$  for denoting the function which interprets the symbol  $f$  in the state  $\lambda$ . Subsets of  $\mathcal{U}$ , called universes, are modeled by unary functions from  $\mathcal{U}$  to  $\mathcal{U}$ ,  $\dots$ . Such a function returns  $\dots$  for all elements belonging to the universe, and  $\dots$  otherwise. A function  $f$  from a universe  $U$  to a universe  $V$  is a unary operation on the superuniverse such that for all  $a \in U$ ,  $f(a) \in V$  and  $f(a) = \text{undef}$  otherwise. The universe  $\dots$  consists of  $\dots$  and  $\dots$ .

A basic ASM  $\dots$  is of the form

$$f(t_1, \dots, t_n) := t_0$$

where  $f(t_1, \dots, t_n)$  and  $t_0$  are closed terms (i.e. terms containing no free variables) in the signature  $\Sigma$ . The semantics of such a rule is this: evaluate all the terms in the given state, and update the function corresponding to  $f$  at the value of the tuple resulting out of evaluating  $(t_1, \dots, t_n)$  to the value obtained by evaluating  $t_0$ . Rules are composed in a parallel fashion, so the corresponding updates are all executed at once. Apart from the basic transition rule shown above, there also exist  $\dots$  rules where the firing depends on the evaluated boolean condition-term,  $\dots$  rules which allow the firing of the same rule for all the elements of a universe, and lastly  $\dots$  rules which are used for introducing new elements into a universe. Transition rules are recursively built up from these rules. Of course not all functions can be updated, for instance the basic arithmetic operations are typically not redefinable.

## 5 Model Transformations

The main motivation behind MDA is to shift the focus of software development from coding to solution modeling by assuming models as the primary artifacts of the development. Key concepts of MDA are model transformations intended as programs which mutates one model into another similarly to a compiler.

In the sequel, unidirectional stateless transformations are given to map Webile models into Conallen and XDW ones. Unidirectional transformations map the source metamodel into the target metamodel but not the converse. Although this may appear a limitation, in practical cases this is essentially unavoidable since a bidirectional transformation implies the adoption of declarative rule-based formalisms which pose severe questions about the termination of transformations. A persistent (in contrast with stateless) model transformation enables change propagation, in the sense that performing the transformation when the source model has changed does not always result in a newly creation model. In fact, persistence implies version policies towards the target model which in combination with information tracking allows not to rewrite completely the target model for different incarnations of the transformation. An interesting and detailed discussion on model transformation languages can be found in [25].

The transformations are defined as ASM rules which starting from an algebra encoding the source model, return an algebra encoding the target model. The signature of an algebra encoding a model is induced by the UML metamodel whose

elements define the sorts of the signature, for instance the class and association elements give place to the  $\text{class}$  and  $\text{association}$  sorts, i.e. the algebra has two universes containing distinguished representatives for all the classes and associations in the model. Stereotypes extending the model elements define subsets in the universes induced by the extended elements itself. This is nicely modeled since ASMs allow subsorting, for instance in the Webile profile the  $\ll\text{DataEntity}\gg$  and  $\ll\text{DataSource}\gg$  stereotypes induces the following subsorting relations

$$\text{class} < \ll\text{DataEntity}\gg \quad \text{and} \quad \text{association} < \ll\text{DataSource}\gg$$

Additionally, the metamodels induce also functions which provide with support to model navigation, e.g. the associations have source and target functions

$$\text{source} : \text{association} \rightarrow \text{class} \quad \text{target} : \text{association} \rightarrow \text{class}$$

which return the source and the target class of the association. Methods are represented by the sort  $\text{method}$  and the class they belong to is computed by the function

$$\text{class} : \text{method} \rightarrow \text{class}$$

further functions defined over methods are  $\text{name}$  and  $\text{body}$  which return the name and the body of a method, respectively. Also tagged values are encoded by means of functions, for example the tagged value *Cardinality* of the  $\ll\text{DataSource}\gg$  stereotyped association defines

$$\text{cardinality} : \ll\text{DataSource}\gg \rightarrow \{ \text{min}, \text{max} \}$$

Moreover, further functions and sorts are given by the basic data types and by those functions which are used in transition rules to accumulate information during the transformation. As an example, the algebraic encoding of the model in Fig. 1 is illustrated in Fig. 5.

In the next sections, the ASM rules for generating the PSMs for the Model and for the View and the Controller are presented, respectively, according to the Fig. 2.

### 5.1 Model Transformation: View-Controller

The transformation introduced here consists of a number of ASM rules, in particular for each structured content the rule  $\text{gen\_client\_server\_pages}$  extends the algebra encoding the source model with three new classes, two server pages modeling the view and the controller and a client page which is generated by the view server page. Furthermore, the rule introduces the following functions

$$\begin{aligned} \text{gen\_client\_server\_pages} : \text{structured\_content} &\rightarrow \text{client\_page} \\ \text{gen\_client\_server\_pages} : \text{structured\_content} &\rightarrow \text{server\_page} \end{aligned}$$

used to track the structured contents from which the client and server pages have been generated. The rule is

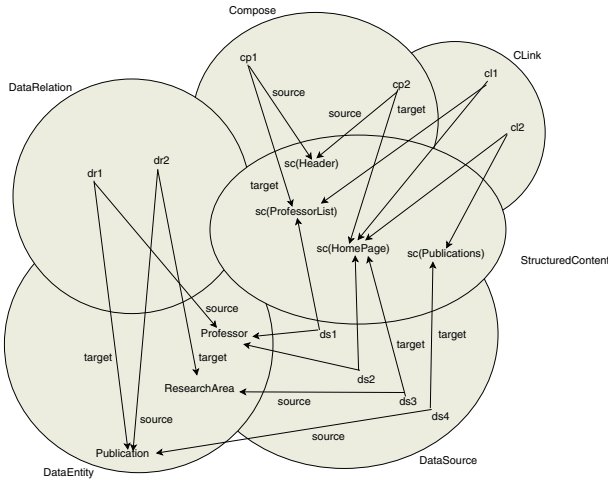


Fig. 5. A model encoded in an algebra

```

asm StructuredContent is
  do forall x in StructuredContent
    extend ServerPage with s1,s2 and ClientPage with c and Build with b
      and Forward with r and Use with u
        source(b) := s1, target(b) := c
        source(r) := s2, target(r) := s1
        source(u) := s2, target(u) := bd
        controller(x) := s2, serverView(x) := s1, clientView(x) := c
    extend Operation with op
      name(op) := "process_request"
      body(op) := Invocations(x)
      belong(op) := s2
    endextend
  endextend
enddo
endasm
  
```

Line (1) in the above rule contains a reference to  $bd$ , the representative of the business delegate component which is incrementally assigned the query operations; line (2) contains the invocations to the  $Invocations$  sub-machine which computes and returns the list of method names which the controllers have to invoke in their body.

The  $CLink$  rule for each  $\llangle CLink \rrangle$  stereotyped association in Webile extends the universe  $U$  with a new element whose source and target are the linked  $ClientPage$  and  $ServerPage$ , respectively

```

asm CLink is
  do forall x in CLink
    extend Link with l
      source(l):=clientView(source(x)), target(l):=controller(target(x))
  
```



```

endextend
enddo
endasm

```

The rules described up to now are not very complex, they could even be considered declarative, since they make use only of the update rule (simpler than in attribute grammars, for instance, which requires some resolution). Algebraically, they can be given as a set of positive conditional equations which induce a (free) functorial transformation on the source algebras. Finally, the rules for handling the composition of structured contents and non-contextual links are missing, since their complexity is comparable to that of the rules above.

### 5.2 Model Transformation: Model

The most interesting rules are not just attributions as the ones above. It is crucial, to be able to collect information while navigating the model, as when computing the transitive closure of a relation for instance. The following rule `DefineAllContents` has to generate the specification of the query operations in the business delegate as relational algebra expressions starting from the data sources in the Webile model. Depending on the tagged value `Label` of the `«DataSource»` associations, the way the contents are retrieved is defined giving place to different expressions. All the `«DataSource»` stereotyped associations related to a specific `«StructuredContent»` can be grouped according to their `Label` tagged value and associated to a `Label`-indexed query operation. The rule has to navigate the source model to understand which data entities are involved in the relational algebra expressions. The `DefineAllContents` rule is defined as follows

```

asm DataSource is
  DefineAllContents
  do forall x in StructuredContent and l in Label : cont(x,l)!=undef
    extend Operation with op
      belong(op) := bd
      name(op) := "get"+name(l)
      choose t in TransfObject : name(t)=name(l)+"TO"
        type(op) :=t
      endchoose
      body(op) := Expr(x,l)
    endextend
  enddo
endasm

```

where `DefineAllContents` sub-machine creates lists of data sources according to the `Label` tagged value partitioning explained above. The rule is given below and makes use of `addListElement` which adds elements to a list

```

asm DefineAllContents is
  do forall x in StructuredContent
    do forall y in DataSource : target(y)=x
      do forall l in Label : label(y)=l
        addListElement(cont(x,l),y)
      enddo
    enddo
  enddo
endasm

```

```

enddo
enddo
endasm
    
```

The sub-machine  $\mathcal{M}_l$  of  $\mathcal{M}_x$  generates the relational algebra expression whose evaluation supplies the content  $l$  for the structured content  $x$ .

```

asm Expr(x, l) is
  extend Body with y
    join(y) := unify(findPath(cont(x,l)))
    selectionKey(y) := findKey(cont(x,l))
  return y
endextend
endasm
    
```

To better understand this rule, let us consider Fig. ?? where an abstract representation of a Webile model is presented. The structured content  $\mathcal{C}$  is fed by three data sources  $ds_1$ ,  $ds_2$  and  $ds_3$  with the same  $\mathcal{C}$  label. In order to obtain a relational algebra expression

$$\sigma_F(T_1 \bowtie_{c_1} T_2 \bowtie_{c_2} T_3 \dots \bowtie_{c_{n-1}} T_n)$$

two macro steps have to be executed:

- the definition of joins between the right relations and,
- the definition of the selection formula  $F$ .

The former is obtained by means of the  $\mathcal{M}_l$  rule, the latter by means of the  $\mathcal{M}_x$  rule. Note that the definition of the expression is not trivial and, due to space limitation, we present the solution by outlining the description for the  $\mathcal{M}_l$ ,  $\mathcal{M}_x$  and  $\mathcal{M}_c$  rules. Two data entities involved in the definition of a content by means of two  $\ll$ DataSource $\gg$  associations, may give place to ambiguous scenarios. In fact,  $E_1$  and  $E_3$  in Fig. 6, are related by means of two different paths. This causes problem for the definition of the joins involving them. Webile deals with this

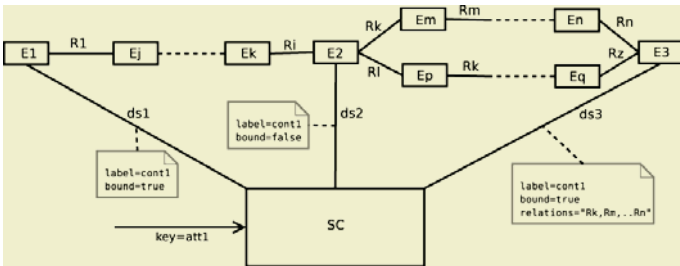


Fig. 6. An abstract representation of a Webile model

problem by means of the tagged value `path` of the `«DataSource»` stereotype. This is used by the `propagatePath` rule which, for each pair of entities involved in the content definition, finds the right path of relations connecting them.

For this rule, the set *Path* is defined as  $DataRelation^* \times Bool$  whose elements are terms  $(R, B)$ , where the first parameter *R* is the list of relations defining the path in the source model, the second parameter *B* is a boolean denoting whether the conditions of the joins involving the entities in the relation chain are empty or equals to conjunctions of equations involving the corresponding keys. For instance, in Fig. 6, `propagatePath` returns the list containing the following elements:  $\{path(R_k, R_m, \dots, R_n, true), path(R_1, \dots, R_i, false)\}$ .

The `propagatePath` rule evaluates the paths and defines the joins between the relations in the paths recursively, whereas `propagateJoin` defines the selection formula for the final expression. Accordingly, if  $E_1$  contained the attribute  $att_1$  in Fig. 6, the formula *F* would be the equation  $E_1.att1 = att1$ . Otherwise, if  $E_2$  contained the attribute  $att_1$ , then the key propagated by the contextual link has no effect and the selection formula is empty. The relational algebra expression defined with this process represents the body of a server-side operation part of the server page obtained by means of the transformation of the structured content.

The last rule handles the creation of the transfer objects, i.e. each query in the business delegate returns a different transfer object type which needs to be defined, as follows

```
asm CreateTransfObj is
do forall l in Label
  extend TransfObject with t
  name(t) := name(l)+"TO"
  do forall d in DataSource : label(d)=l
    do forall a in DataAttribute : belong(a)=source(d)
      extend Attribute with a1
      name(a1) := name(a), type(a1) := type(a)
    endextend
  enddo
enddo
endextend
enddo
endasm
```

## 6 Related Work

Model transformations are increasingly gaining attention in different areas of software design, development and integration. Such significance is also witnessed by the OMG's Queries/Views/Transformations RFP [16] issued to define a standard way of performing model transformations. The submitted proposals range from imperative unidirectional stateless transformations [23] to declarative persistent bidirectional ones [9]. The former does not allow any form of persistence, while the latter may easily lead to solutions which can take potentially unbounded time to execute [25]. Other approaches are considered hybrid, such

as ATL [4], since they wrap imperative bodies inside declarative shells to specify unidirectional transformations.

Focusing on the languages for describing Web applications, many topics may be related to the work proposed here, among them many are suggesting extensions to model web applications using UML. Apart from Conallen's work, it is worth mentioning the approaches proposed by Koch and Hennicker [17]. In particular, they structure the design of web applications into three different aspects: content, navigational structure and presentation (the data dimension is not considered). Since these aspects are related together, they propose to model each one using a different UML model and to relate them together using mapping rules. This separation (present also in other approaches [8, 20]) is currently missing in Webile. Differently from us, Koch and Hennicker use UML class and sequence diagrams to describe and model behavior.

Regarding model-based code generation for the Web, Kraus and Koch [19] show how the UML design models produced in [17] can be automatically mapped into XML documents. Araneus [20] and WebML [8] represent the more interesting model-based approaches for the Web, especially WebML which is supported by the WebRatio commercial tool.

## 7 Conclusions and Future Work

This paper describes a model-driven approach for the development of data-intensive Web applications. Starting from conceptual models that do not refer to any technological asset, formal model transformations are used to obtain several PSMs for different aspects of an MVC conformant J2EE application. Compared with techniques which allow . . . , model-to-code generation, flexible and practical model transformations enhance traceability and consistency between models and code, since they tend to diverge as soon as changes are manually operated on the generated applications.

Model transformations are intrinsically difficult. According to our experience they should combine desirable features as formality and declarativeness with good pragmatic qualities. The transformations introduced above are applicable in a declarative way since models are queried by means of first-order predicates and subsequently manipulated in an operational fashion.

Model composition is also important, since generating models from the same source model requires to merge them on certain correspondences. Exploiting algebraic and categorical constructions (in the sense of category theory), which have been investigated since decades, we plan to define a theory of transformation composition to perform complex model weaving and transformation maintenance.

## Acknowledgments

We thank Amleto Di Salle and Fabio Mancinelli for the lively and enlightening discussions. Also, we are grateful for the insightful comments we received from the anonymous reviewers.

## References

1. D. Alur, J. Crupi, and D. Malks. *Core J2EE Patterns*. Sun Microsystems Press (Prentice Hall), 2nd edition, 2003.
2. M. Anlauff. XASM – An Extensible, Component-Based Abstract State Machines Language. In *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 69–90. Springer, 2000.
3. M. Anlauff, S. Chakraborty, P. Kutter, A. Pierantonio, and L. Thiele. Generating an action notation environment from Montages descriptions. *Int. J. Software Tools for Technology Transfer*, 3(4):431–455, 2001.
4. J. Bézivin, G. Dupé, F. Jouault, G. Pitette, and J.E. Rougui. First Experiments with the ATL model transformation language: Transforming XSLT into XQuery. In *2nd OOPSLA W. Generative Techniques in the context of MDA*, 2003.
5. E. Börger. Why Use Evolving Algebras for Hardware and Software Engineering? In *Procs. SOFSEM '95, 22nd Seminar on Current Trends in Theory and Practice of Informatics*, volume 1012 of *LNCS*, pages 236–271. Springer, 1995.
6. E. Börger. The Origins and the Development of the ASM Method for High Level System Design and Analysis. *J. Universal Computer Science*, 8(1):2–74, 2002.
7. E. Börger and R. Stärk. *Abstract State Machines - A Method for High-Level System Design and Analysis*. Springer, 2003.
8. S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a Modeling Language for Designing Web sites. *Computer Networks*, 33(1–6):137–157, 2000.
9. Compuware and Sun. XMOF queries, views and transformations on models using MOF, OCL and patterns, 2003. OMG Document ad/2003-08-07.
10. J. Conallen. Modeling Web Application Architectures with UML. *Comm. ACM*, 42(10):63–71, 1999.
11. Enterprise JavaBeans. <http://java.sun.com/products/ejb/>.
12. E. Romina. Modellazione concettuale di un portale mediante UML. Master's thesis, Università degli Studi dell'Aquila, 2003/04. In italian.
13. P. Fraternali. Tools and Approaches for Developing data-intensive Web Applications: A Survey. *ACM Computing Surveys*, 31(3):227–263, 1999.
14. S.T. Pope G.E. Krasner. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *J. Object-Oriented Programming*, 1(3):26–49, 1988.
15. Object Management Group. OMG/Model Driven Architecture - A Technical Perspective, 2001. OMG Document: ormsc/01-07-01.
16. Object Management Group. MOF 2.0 Query/Views/Transformations RFP, 2002. OMG document ad/02-04-10.
17. R. Hennicker and N. Koch. Systematic Design of Web Applications with UML. In *Unified Modeling Language: Systems Analysis, Design and Development Issues*, chapter 1, pages 1–20. Idea Publishing Group, 2001.
18. Java Data Objects. <http://java.sun.com/products/jdo/>.
19. A. Kraus and N. Koch. Generation of Web Applications from UML Models using an XML Publishing Framework. In *Procs. 6th World Conference on Integrated Design and Process Technology (IDPT)*, volume 1, 2002.
20. P. Merialdo, P. Atzeni, and G. Mecca. Design and Development of data-intensive Web sites: The Araneus approach. *ACM Trans. on Internet Technology*, 3(1):49–92, 2003.
21. P.-A. Muller, P. Studer, and J. Bezivin. Platform independent web application modeling. In *UML 2003*, volume 2863 of *LNCS*, pages 220–233. Springer, 2003.

22. A. Vallecillo N. Moreno. Using MDA for Designing and Implementing Web-based Applications. In *Int. Conf. on Web Engineering*, Munich, Germany, 2004. Tutorial.
23. OpenQVT. Response to the MOF 2.0 Queries / Views / Transformations RFP, 2003. OMG Document ad/2003-08-05.
24. D. Di Ruscio, H. Muccini, and A. Pierantonio. A Data Modeling Approach to Web Application Synthesis. *Int. J. Web Engineering and Technology*, 1(3):320–337, 2004.
25. L. Tratt. Model transformations and tool integration. *J. Software and Systems Modeling*, 2004. To appear.

# Automated Reasoning on Feature Models<sup>\*</sup>

David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés

Dpto. de Lenguajes y Sistemas Informáticos,  
University of Seville, Av. de la Reina Mercedes S/N, 41012 Seville, Spain  
{benavides, trinidad, aruiz}@tdg.lsi.us.es

**Abstract.** Software Product Line (SPL) Engineering has proved to be an effective method for software production. However, in the SPL community it is well recognized that variability in SPLs is increasing by the thousands. Hence, an automatic support is needed to deal with variability in SPL. Most of the current proposals for automatic reasoning on SPL are not devised to cope with extra-functional features. In this paper we introduce a proposal to model and reason on an SPL using constraint programming. We take into account functional and extra-functional features, improve current proposals and present a running, yet feasible implementation.

## 1 Introduction and Motivation

Research on SPLs is thriving. Unlike other approaches reuse in SPL has to become systematic instead of ad-hoc. In order to achieve such a goal, SPL practices guide organizations towards the development of products from existing assets rather than the development of separated products one by one from scratch. Thus, features that are shared by all SPL products are reused in every single product. Most of the existing methods [3, 6] for SPL engineering agree that a way for modelling SPL is needed. In this context feature models [7, 9, 11, 13, 22] have been quoted as one of the most important contributions to SPL modelling [7, pag.82]. As in other cases, first applications in routine production are stimulating the development of a supporting science for improving the production methods [17].

Feature models are used to model SPL in terms of features and relations amongst them. In this type of models, the number of potential products of an SPL increases with the number of features. Consequently, a large number of features lead to SPLs with a large number of potential products. In an extremely flexible SPL, where all features may or may not appear in all potential products, the number of potential products is equal to  $2^n$ , being  $n$  the number of features. Moreover, current feature models are only focused on modelling functional features and in the most quoted proposals [7, 9, 11, 13, 22] there is a lack of modelling artifacts that deal with extra-functional features (features related to so-called quality or non-functional features). If extra-functional features are taken

---

<sup>\*</sup> A preliminary version of this paper was presented at [4]. This work was partially funded by the Spanish Ministry of Science and Technology under grant TIC2003-02737-C02-01 (AgilWeb) and PRO-45-2003 (FAMILIES).

into account the number of potential products increases even further. Although it is accepted that in an SPL it is necessary to deal with these extra-functional features [5, 11, 12], there is no consensus about how to deal with them.

Automated reasoning is an ever challenging field in SPL engineering [18, 23]. It should be considered specially when the number of features increases due to the increase in the number of potential products. To the best of our knowledge, there are only a couple of limited attempts by Van Deursen *et al.* and Mannion [8, 14] that treat automatic manipulation of feature models. Although those proposals only consider functional features, leaving out extra-functional features. Van Deursen *et al.* [8] explore automated manipulation of feature descriptions providing an algebra to operate on the feature diagrams proposed in [7]. Mannion's proposal [14] uses first-order logic for product line reasoning. However it only provides a model based on propositional-logic using *AND*, *OR* and *XOR* logical operators to model SPLs. Both attempts have several limitations:

1. They do not allow to deal with extra-functional features (both attempts leave this work pending).
2. They basically answer to the single question of how many products a model has.
3. As far as we know, they have no available an implementation.

In addition, Mannion's model uses the *XOR* ( $\oplus$ ) operator to model alternative relations, which is either a mistake or a limitation because the model becomes invalid if more than two features are involved in an alternative relation.

The contribution of this paper is threefold. First, we extend existing feature models to deal with extra-functional features. Secondly, we deal with automatic reasoning on extended feature models answering five generic questions, namely *i*) how many potential products a model has *ii*) which is the resulting model after applying a filter (e.g. users constraint) to a model, *iii*) which are the products of a model, *iv*) is it a valid model, and *v*) which is the best product of a model according to a criterion and finally giving an accessible, running implementation.

The remainder of this paper is structured as follows. In Section 2, we propose an extension to deal with extra-functional features. In Section 3, we present a mapping to transform an extended feature model into a Constraint Satisfaction Problem (CSP) in order to formalize extended feature models using constraint programming [15]. In Section 4, we improve current reasoning on feature models and we give some definitions to be able to automatically answer several questions on extended feature models. In Section 5, we show how our model can be applied to other important activities such as obtaining commonality and variability information. In Section 6, we briefly present a running prototype implementation. Finally, we summarize our conclusions and describe our future work in Section 7.

## 2 Extending Feature Models with Extra-Functional Features

### 2.1 Feature Models

The main goal of feature modelling is to identify commonalities and differences among all products of a SPL. The output of this activity is a compact representation of all po-



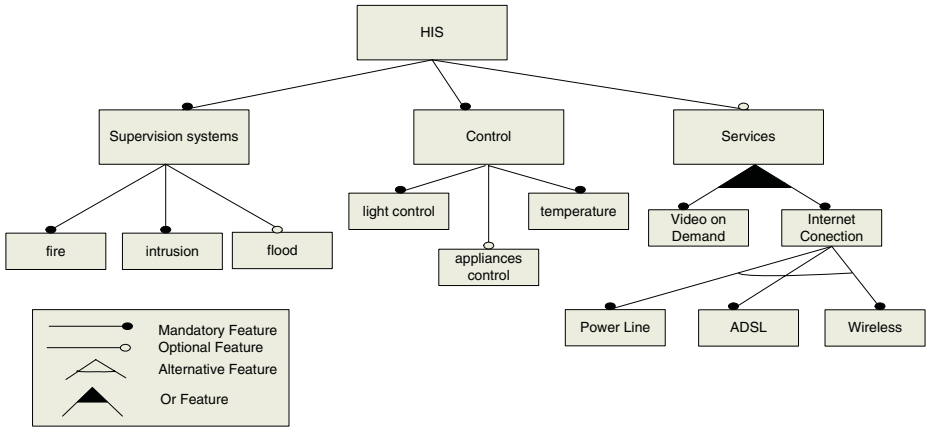


Fig. 1. Feature model of an SPL in the HIS domain

tential products of an SPL, hereinafter called "*feature model*". Feature models are used to model SPL in terms of features and relations among them. Roughly speaking, a feature is a distinctive characteristic of a product. Depending on the stage of development, it may refer to a requirement [10], a component in an architecture [2] or even to pieces of code [16] (feature oriented programming) of a SPL.

There are several notations to design feature models [7, 9, 11, 13, 22]. We found that the one proposed by Czarnecki is the most comprehensible and flexible as well as being one of the most quoted. Figure 1 depicts a possible feature model of an SPL for the domain of Home Integration Systems (HIS) using Czarnecky's notation. This example is partially inspired by [13].

Czarnecki's notation proposes four relations, namely: mandatory, optional, alternative and or-relation. In these relations, there is always a parent feature and one (in the case of mandatory and optional relations) or more (in the case of alternative and or-relation) child features.

- **Mandatory**: the child feature in this relation is always present in the SPL's products when its parent feature is present. For example, Every HIS is equipped with *i*) fire and intrusion supervision systems and *ii*) light and temperature control.
- **Optional**: the child feature in an optional relation may or may not be present in a product when its parent feature is present. For Example, there are HISs with services and others without them.
- **Alternative**: a child feature in an alternative relation may be present in a product if its parent feature is included. In this case, only one feature of the set of children is present. For example, in a HIS product if an Internet connection is included, then the customer has to choose between an ADSL, powerline or wireless connection, but only one.
- **Or-relation**: the child feature in an or-relation may be present in a product if its parent feature is included. Then, at least one feature of the set of children may be present. For example, in a HIS the products may have Video or Internet or both at the same time.

This model includes 32 potential products (you can check this on <http://www.tdg-seville.info/topics/spl>). Examples of them are: *i*) Basic product: consisting of a fire and intrusion supervision systems and light and temperature control. *ii*) Full product: a product with all supervision and control features as well as a power line, ADSL or wireless Internet connection.

## 2.2 Extended Feature Models

Current proposals only deal with characteristics related to the functionality offered by an SPL (functional features). Thus, there exists no solid proposal for dealing with the remaining characteristics, also called extra-functional features. There are several concepts that we would like to clarify before analyzing current proposals and framing our contribution:

- Feature: a prominent characteristic of a product. Depending on the stage of development, it may refer to a requirement [10] (if products are requirement documents), a component in an architecture [2] (if products are component architectures) or even to pieces of code [16] (if products are binary code in a feature oriented programming approach) of an SPL.
- Attribute: the attribute of a feature is any characteristic of a feature that can be measured. *Availability* and *cost* are examples of attributes of the *Service* feature of figure 1. *Latency* and *bandwidth* may be examples of attributes of an Internet connection.
- Attribute domain: the space of possible values where the attribute takes its values. Every attribute belongs to a domain. It is possible to have discrete domains (e.g.: integers, booleans, enumerated) or continuous domains (e.g.: real).
- Extra-functional feature: a relation between one or more attributes of a feature. For instance:  $bandwidth = 256, Latency/Availability > 50$  and so on. These relations are associated to a feature.

In figure 1, every feature refers to functional features of the HIS product line so that every product differs because of its functional features. However, every feature of figure 1 may have associated extra-functional features. For instance, considering the *services* feature, it is possible to identify extra-functional features related to it, such as relations among attributes like *availability*, *reliability*, *development time*, *cost* and so forth. Likewise the *Internet Connection* feature can have extra-functional features such as relations among *latency* or *bandwidth*. Furthermore, the attributes' values of extra-functional features can differ from one product to another. It means, every product not only differs because of its functional features, but because of its extra-functional features too.

Consider the full product of the HIS product line example presented formerly with the same functional features. It is possible to offer several products with the same functional features but different extra-functional features, for instance: *i*) High quality full product: a product with full functionality and high quality: high *availability* and *reliability* and high *cost* too. *ii*) Basic quality full product: a product with full functionality but lower quality: lower *availability* and *reliability* and lower *cost* too.

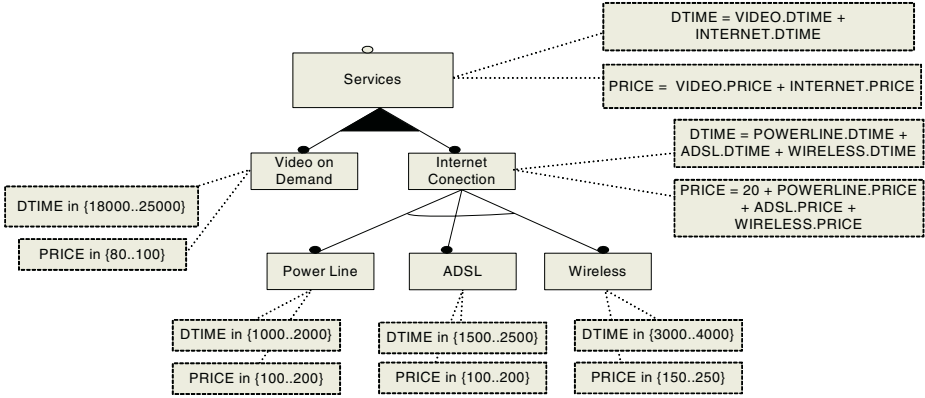


Fig. 2. Extended feature model for an SPL in the HIS domain

To date, we have not found any proposal dealing with functional and extra-functional features in the same model. However, there are some works in the literature that suggest the need of dealing with extra-functional features: Kang *et. al* have been suggesting the need to take into account extra-functional features since 1990 [11, pag. 38] when they depicted a classification of features, although they did not provide a way to do it. Later, in 1998 Kang *et. al* [12] made an explicit reference to what they called 'non-functional' features (a possible type of what we call extra-functional features). However the authors still did not propose a way to solve it. In 2001 Kang *et. al* [5], proposed some guidelines for feature modelling: in [5, pag. 19], the authors once again made the distinction between functional and quality features and pointed out the need of a specific method to include extra-functional features, but they did not provide this specific method on this occasion either.

### 2.3 A Notation for Extended Feature Models

We propose to extend Czarnecki's feature models with extra-functional features and improve previous vague notations proposed in [20] by allowing relations amongst attributes. Using the HIS example, every feature may have one or more attribute relations, for example, the *price* ( $PRICE$ ) and *development time* (expressed in hours) ( $DTIME$ ) taking a range of values in both a discrete or continuous domain (integer or real for example). Thus, it would be possible to decorate the graphical feature model with this kind of information. Figure 2 illustrates a piece of the feature model of figure 1 with extra-functional features with our own notation inspired by [20].

In this example, every sub feature of the *Service* feature has two attributes:  $PRICE$  and  $DTIME$ . Each of the attributes of leaf features are in a domain of values. For instance, the price of an ADSL connection can range from 100 to 200<sup>1</sup>. In the case of parent features, the values of the attributes are the addition of their children values. For example, the price of an Internet connection is the sum of the prices of the possible Internet connections.

<sup>1</sup> These values are just illustrative, they may have nothing to do with real values.

### 3 Mapping Extended Feature Models onto CSP

#### 3.1 Preliminaries

Constraint Satisfaction Problems [21] have been object of research in Artificial Intelligence in the last few decades. A Constraint Satisfaction Problem (CSP) is defined as a set of variables, each ranging on a finite domain, and a set of constraints restricting all the values that variables can take simultaneously. A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that all constraints are satisfied simultaneously. We may want to find: *i*) just one solution, with no preference as to which one, *ii*) all solutions, *iii*) an optimal solution by means of an objective function defined in terms of one or more variables. Solutions to a CSP can be found by searching (systematically) through all possible value assignments to variables.

In many real-life applications, we do not want to find any solution but a good solution. The quality of a solution is usually measured by an application dependent function called objective function. The goal is to find a solution that satisfies all the constraints and minimize or maximize the objective function, respectively. Such problems are referred to as Constraint Satisfaction Optimization Problems (CSOP), which consist of a standard CSP and an optimization function that maps every solution (complete labelling of variables) to a numerical value. These are some basic definitions of what a CSP is.

**Definition 1 (CSP).** A CSP is a three-tuple of the form  $(V, D, C)$  where  $V \neq \emptyset$  is a finite set of variables,  $D \neq \emptyset$  is a finite set of domains (one for each variable) and  $C$  is a constraint defined on  $V$ .

Consider, for instance, the CSP:  $(\{a, b\}, \{[0..2], [0..2]\}, \{a + b < 4\})$

**Definition 2 (Solution).** Let  $\psi$  be a CSP, a solution of  $\psi$  is whatever valid assignment of all elements in  $V$  as satisfies  $C$ .

In the previous example, a possible solution is  $(2, 0)$  since it verifies that  $2 + 0 < 4$ .

**Definition 3 (Solution space).** Let  $\psi$  be a CSP of the form  $(V, D, C)$ , its solution space denoted as  $sol(\psi)$  is made up of all its possible solutions. A CSP is satisfiable if its solution space is not empty.

$$sol(\psi) = \{S \mid \forall s_i \cdot s_i \in S \Rightarrow C(s_i) = true\}$$

In the previous example, there are eight solutions. The only assignment that does not satisfy  $a + b < 4$  is  $(2, 2)$ . Nevertheless, if we replace the constraint with  $a + b < -1$ , then the CSP is not satisfiable.

**Definition 4 (CSOP).** A CSOP is a four-tuple of the form  $(V, D, C, O)$  where  $V$ ,  $D$  and  $C$  stand for a CSP and  $O$  is a real function defined on  $D$ .

Consider, for instance, the CSOP:  $(\{a, b\}, \{[0..2], [0..2]\}, \{a + b < 4\}, a)$

**Definition 5 (Optimum space).** Let  $\psi$  be a CSOP of the form  $(V, D, C, O)$ , its optimum space denoted as  $max/min(\psi, O)$  is made up of all solutions that maximize or minimize  $O$ .

$$max(\psi, O) = \{s \mid \forall st \cdot st \in sol(\psi) \wedge st \neq s \Rightarrow O(s) \geq O(st)\}$$

$$\min(\psi, O) = \{s \mid \forall st \cdot st \in \text{sol}(\psi) \wedge st \neq s \Rightarrow O(s) \leq O(st)\}$$

In the previous example,  $\max(\psi, a) = \{(2, 0), (2, 1)\}$ .

### 3.2 The Mapping

In [1] we presented an algorithm to transform an extended feature model into a CSP. The mapping between a feature model and a CSP has the following general form: *i*) the features make up the set of variables, *ii*) the domain of each variable is the same:  $\{true, false\}$ , *iii*) extra-functional features are expressed as constraints and *iv*) every relation of the feature model becomes a constraint among its features in the following way:

- Mandatory relation: Let  $f$  be the parent and  $f_1$  the child in a mandatory relation , then the equivalent constraint is:  $f_1 = f$
- Optional relation: Let  $f$  be the parent and  $f_1$  the child in an optional relation, then the equivalent constraint is:  $f_1 \Rightarrow f$
- Or-relation: Let  $f$  be the parent in an or-relation and  $f_i \mid i \in [1 \dots n]$  the set of children, then the equivalent constraint is:  $f_1 \vee f_2 \vee \dots \vee f_n \Leftrightarrow f$ .
- Alternative relation: Let  $f$  be the parent of an alternative relation and  $f_i \mid i \in [1 \dots n]$  the set of children, then the equivalent constraint is:  
 $(f_1 \Leftrightarrow (\neg f_2 \wedge \dots \wedge \neg f_n \wedge f)) \wedge (f_2 \Leftrightarrow (\neg f_1 \wedge \neg f_3 \dots \wedge \neg f_n \wedge f)) \wedge$   
 $(f_n \Leftrightarrow (\neg f_1 \wedge \dots \wedge \neg f_{n-1} \wedge f))$

There may be several different algorithms to map extended feature models. The one presented in [1] is a possible one. Hereinafter, we refer to the equivalent CSP resulting from the mapping as  $\psi_M$ . Using this mapping, constraints for functional and extra-functional features can be handled together. Thus, table 1 shows the equivalent constraints for figure 1 with the extra-functional features of figure 2. Constraints of extra-functional features are denoted by an asterisk. *POWERLINE*, *ADSL* and *WIRELESS* extra-functional features are not shown for lack of space as they are very similar to the *VIDEO*. ones.

## 4 Automated Reasoning on Extended Feature Models

Since we go toward automated reasoning on feature models, a formal model of SPL becomes necessary. We propose to use Constraint Programming to reason on extended features models.

Our model is able to answer the following questions:

### 4.1 Number of Products

One of the questions to be answered is how many potential products a FM contains. This is a key question in SPL engineering because if the number of products increases the SPL becomes more flexible as well as more complex.

**Table 1.** A trace of the algorithm presented in [1] for HIS example

Relation	$\psi_{HIS}$
HIS 1	$(SUPERVISION = HIS)$
HIS 2	$(CONTROL = HIS)$
HIS 3	$(SERVICES \Rightarrow HIS)$
SUPERVISION 1	$(FIRE = SUPERVISION)$
SUPERVISION 2	$(INTRUSION = SUPERVISION)$
SUPERVISION 3	$(FLOOD \Rightarrow SUPERVISION)$
CONTROL 1	$(LIGHT = CONTROL)$
CONTROL 2	$(APPLIANCE \Rightarrow CONTROL)$
CONTROL 3	$(TEMPERATURE = CONTROL)$
SERVICES 1	$((VIDEO \vee INTERNET) \Leftrightarrow SERVICES)$
SERVICES *	$(SERVICES.PRICE = VIDEO.PRICE + INTERNET.PRICE) \wedge$ $(SERVICES.DTIME = VIDEO.DTIME + INTERNET.DTIME)$
VIDEO *	$((VIDEO.PRICE \in [80\ 100]) \Leftrightarrow VIDEO) \wedge$ $((VIDEO.PRICE = 0) \Leftrightarrow \neg VIDEO) \wedge$ $((VIDEO.DTIME \in [18000, 25000]) \Leftrightarrow VIDEO) \wedge$ $((VIDEO.DTIME = 0) \Leftrightarrow \neg VIDEO)$
INTERNET 1	$(POWERLINE \Leftrightarrow (\neg ADSL \wedge \neg WIRELESS \wedge INTERNET)) \wedge$ $(ADSL \Leftrightarrow (\neg POWERLINE \wedge \neg WIRELESS \wedge INTERNET)) \wedge$ $(WIRELESS \Leftrightarrow (\neg POWERLINE \wedge \neg ADSL \wedge INTERNET))$
INTERNET *	$((INTERNET.PRICE = ADSL.PRICE + WIRELESS.PRICE$ $+ POWERLINE.PRICE + 20) \Leftrightarrow INTERNET) \wedge$ $((INTERNET.PRICE = 0) \Leftrightarrow \neg INTERNET) \wedge$ $((INTERNET.DTIME = ADSL.DTIME + WIRELESS.DTIME$ $+ POWERLINE.DTIME) \Leftrightarrow INTERNET) \wedge$ $((INTERNET.DTIME = 0) \Leftrightarrow \neg INTERNET)$

**Definition 6 (Cardinal).** Let  $M$  be an extended feature model, the number of potential products of  $M$ , hereinafter cardinal, is equal to the solution number of its equivalent CSP  $\psi_M$ .

$$cardinal(M) = |sol(\psi_M)|$$

In the HIS example of figure 1  $cardinal(HIS) = 32$ , simply by adding for example a new service like *Radio Streaming*, the number of potential products raises to 64. Likewise adding the attributes of figure 2  $cardinal(HIS) = 260$ .

## 4.2 Filter

There should be a way to apply filters to the model. These filters can be imposed by the users. A filter acts as a limitation for the potential products of the model. A typical application of this operation occurs when customers are looking for a product with a specific set of characteristics, that is, they are not interested in all potential products but in some of them only (those passing the filter).

**Definition 7 (Filter).** Let  $M$  be an extended feature model and  $F$  a constraint representing a filter, the filtered model of  $\psi_M$ , hereinafter filter, is equal to  $\psi_M$  adding the constraint  $F$ .

$$\text{filter}(M, F) = (\psi_M \wedge F)$$

A possible filter for the HIS example would be to ask for all products with video on demand, making the number of potential products decrease from 32 to 16. It is also possible to apply filters to attributes. For example, it would be possible to ask for all products whose prices are lower than 200, 12 then

$$\text{cardinal}(\text{filter}(\text{HIS}, \text{SERVICES.PRICE} < 200)) = 44$$

(when any filter is imposed, it decreases from 260 to 44).

### 4.3 Products

Once  $\psi_M$  is defined, there should be a way to get the solutions of the model, that is the products of  $\psi_M$ .

**Definition 8 (Products).** Let  $M$  be an extended feature model, the potential products of the model  $M$ , hereinafter products, is equal to the solutions of the equivalent CSP  $\psi_M$ .

$$\text{products}(M) = \{s \in \text{sol}(\psi_M)\}$$

In the HIS example we would want to get all the possible products of the model or even apply a filter and then get the products. Thus  $M = \text{filter}(\text{HIS}, \text{VIDEO} = \text{true})$  and  $\text{products}(M) = \{s \in \text{sol}(\psi_{\text{HIS}} \wedge \text{VIDEO} = \text{true})\}$ .

### 4.4 Validation

A valid extended feature model is a model where at least one product can be selected. That is, a model where  $\psi_M$  has at least one solution.

**Definition 9 (Valid model).** A feature model  $M$  is valid if its equivalent CSP is satisfiable.

$$\text{valid}(M) \iff \text{products}(M) \neq \emptyset$$

The HIS model of the example is valid, but there might be situations where the constraints are not satisfiable, making the model invalid. For instance, if the Service's price is lower than 100, and a filter is imposed to have *INTERNET*, then the model is not valid:

$$\text{valid}(\text{filter}(\text{HIS}, \text{INTERNET} = \text{true} \wedge \text{SERVICE.PRICE} < 100)) = \text{false}$$

### 4.5 Optimum Products

Finding out the best products according to a determinate criterion is an essential task in our model.

**Definition 10 (Optimum).** *Let  $M$  be an extended feature model and  $O$  an objective function, then the optimum set of products, hereinafter  $\max$  and  $\min$ , is equal to the optimum space of  $\psi_M$ .*

$$\begin{aligned} \max(M, O) &= \max(\psi_M, O) \\ \min(M, O) &= \min(\psi_M, O) \end{aligned}$$

It is also possible to apply a filter to the HIS example and then ask for an optimal product. Thus, a possible optimum criterion for the HIS example would be to ask for all products with video on demand, and the minimum value for the multiplication of price and development time. In this case selected products  $P_{opt}$  are:

The model presented in this section can support current feature models. The only dif-

$$\left. \begin{aligned} M &= \text{filter}(HIS, VIDEO = true) \\ O &= SERVICE.PRICE * SERVICE.DTIME \\ P_{opt} &= \min(M, O) \end{aligned} \right|$$

ference is that current feature models do not support extra-functional features which means that when using our model to reason on current feature models, attributes are not taken into account. Thus, the algorithm presented in [1] and all previous definitions remain valid for current feature models.

## 5 Realising the Benefits

Compared to others, our approach is very flexible because it is so easy to extend. Below, we show two more definitions based on the previous ones to demonstrate how our approach can be extended and give valuable information to SPL engineers.

### 5.1 Variability

As mentioned previously, feature models are composed of a set of features and relations among them. If relations restrict the number of products to only one, we are considering the lowest variability while a feature model defining no possible product would be considered a non-valid model. On the other hand, considering no relations, the number of products within the feature model would be the highest. This case would represent the highest variability. Relations restrict the number of potential products, so variability depends on relation types.

Let a leaf feature be a feature that has no child feature. Parent features add no variability to the model, because they are feature aggregates. We define the variability factor as follows.

**Definition 11 (Variability Factor(VF)).** *Let  $M$  be an extended feature model, and  $\psi_M$  the equivalent CSP. Let  $M^V$  be another extended feature model, considering the leaf features in  $M$  and no relation among its features, and  $\psi_M^V$  the equivalent CSP.*

$$VF(M) = \frac{\text{cardinal}(M)}{\text{cardinal}(M^v)} = \frac{|\text{sol}(\psi_M)|}{|\text{sol}(\psi_M^V)|}$$



The variability factor in the real domain would take values ranging from 0 to 1.

VF can assist decision making. For instance, when many products are going to be developed one of the first decisions to be taken, is whether the SPL approach or traditional approach is going to be applied. A high VF may suggest an SPL approach; a low VF may suggest a traditional approach.

## 5.2 Commonality

In a feature model, some features will appear in every product, some in only one product and others in some products. When deciding the order in which features are going to be developed, it is very important to know which are the most common features in order to prioritize their building. Obtaining commonality information from the feature model can be feasible by asking questions to our model. We define the feature commonality as the percentage of products containing that feature.

**Definition 12 (Commonality).** *Let  $M$  be an extended feature model and  $F$  the feature we want to know its commonality.*

$$\text{commonality}(M, F) = \frac{\text{cardinal}(\text{filter}(M, F = \text{true}))}{\text{cardinal}(M)}$$

## 6 Implementation

We have already implemented some of the ideas presented in this paper using OPL Studio, a commercial CSP solver. This implementation is available at <http://www.tdg-seville.info/topics/spl>.

Three modules have been developed in our implementation: first, a feature markup language and XML Schema were agreed on. This language allows to represent the Czarnecki's feature model [7]. Secondly, a parser to transform this XML documents to a CSP following the algorithm described in [1] was developed. Finally, a web-based prototyping interface was made available to allow to test some of the capabilities of the model. In order to test our implementation, we have modeled four problems (two academical and two real product lines) that are available on the web site.

In order to evaluate the implementation, we measured its performance and effectiveness. We implemented the solution using Java. We ran our tests on a WINDOWS XP PROFESSIONAL machine that was equipped with a 1.5Ghz AMD Athlon XP microprocessor, and 496 MB of DDR 266Mhz RAM memory. The test was based on the feature model in Figure 1, adding new features. Several tests were made on each feature model in order to avoid as many exogenous interferences as possible.

We have experimentally inferred that the implementation presented has an exponential behavior while increasing the number of features in the feature model and maintaining a constant variability factor. We have measured the solving time for  $\text{products}(M)$ , which is the most complex to obtain, and have considered it for different values of VF as shown in Figure 3. Our test determines our model has a good performance up to 25 features while the VF is kept constant.

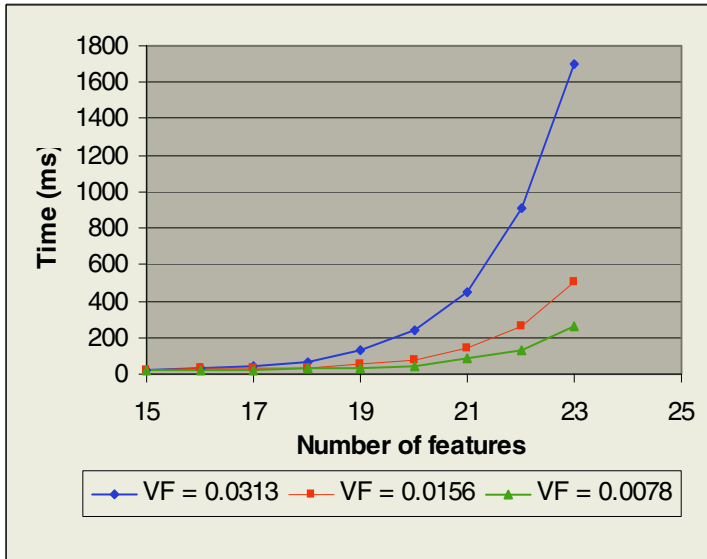


Fig. 3. Empirical performance test for  $products(M)$

## 7 Conclusion and Further Work

In this paper we set the basis for reasoning on SPL with features and attribute relations at the same time and in the same model using constraint programming.

There are some challenges we have to face in the near future, namely: *i*) extending our model to support dependencies such as a feature that *requires* or *excludes* another feature (e.g. video on demand requires *ADSL128*) that are also proposed in other feature models *ii*) extending our current feature markup language to include extra-functional features *iii*) developing a case tool to validate our model on an industrial context, *iv*) performing a more rigorous validation of our implementation, studying the influences as well as the number of solutions, the types of relations, the number of features, and so on, *v*) comparing our work with others in the product configuration field[19].

## References

1. D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Coping with automatic reasoning on software product lines. In *Proceedings of the 2nd Groningen Workshop on Software Variability Management*, November 2004.
2. M. Bernardo, P. Ciancarini, and L. Donatiello. Architecting families of software systems with process algebras. *ACM Transactions on Software Engineering and Methodology*, 11(4):386–426, 2002.
3. J. Bosch. *Design and Use of Software Architectures*. Addison-Wesley, 1<sup>th</sup> edition, 2000.

4. J. Bosch and H. Obbink. Proceedings of the 2nd Groningen Workshop on Software Variability Management. Technical Report to be published, University of Groningen, November 2004.
5. G. Chastek, P. Donohoe, K.C. Kang, and S. Thiel. Product Line Analysis: A Practical Introduction. Technical Report CMU/SEI-2001-TR-001, Software Engineering Institute, Carnegie Mellon University, June 2001.
6. P.C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, August 2001.
7. K. Czarnecki and U.W. Eisenecker. *Generative Programming: Methods, Techniques, and Applications*. Addison–Wesley, may 2000. ISBN 0–201–30977–7.
8. A. van Deursen and P. Klint. Domain–specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002.
9. M. Griss, J. Favaro, and M. d’Alessandro. Integrating feature modeling with the RSEB. In *Proceedings of the Fifth International Conference on Software Reuse*, pages 76–85, Canada, 1998.
10. S. Jarzabek, Wai Chun Ong, and Hongyu Zhang. Handling variant requirements in domain modeling. *The Journal of Systems and Software*, 68(3):171–182, 2003.
11. K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.
12. K.C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A feature–oriented reuse method with domain–specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.
13. K.C. Kang, J. Lee, and P. Donohoe. Feature–Oriented Product Line Engineering. *IEEE Software*, 19(4):58–65, July/August 2002.
14. M. Mannion. Using First-Order Logic for Product Line Model Validation. In *Proceedings of the Second Software Product Line Conference (SPLC2)*, LNCS 2379, pages 176–187, San Diego, CA, 2002. Springer.
15. K. Marriot and P.J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.
16. Christian Prehofer. Feature-oriented programming: A new way of object composition. *Concurrency and Computation: Practice and Experience*, 13(6):465–501, 2001.
17. M. Shaw. Prospects for an engineering discipline of software. *IEEE Softw.*, 7(6):15–24, 1990.
18. M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. COVAMOF: A Framework for Modeling Variability in Software Product Families. In *Proceedings of the Third Software Product Line Conference (SPLC04)*, San Diego, CA, 2004.
19. T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen. Towards a general ontology of configuration. *AI EDAM*, 12(4):357–72, 1998.
20. D. Streitferdt, M. Riebisch, and I. Philippow. Details of formalized relations in feature models using ocl. In *Proceedings of 10th IEEE International Conference on Engineering of Computer–Based Systems (ECBS 2003)*, Huntsville, USA. *IEEE Computer Society*, pages 45–54, 2003.
21. Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1995.
22. J. van Gorp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA’01)*, *IEEE Computer Society*, pages 45–54, 2001.
23. A. Wasowski. Automatic Generation of Program Families by Model Restrictions. In *Proceedings of the Third Software Product Line Conference (SPLC04)*, San Diego, CA, 2004.

# A Method for Information Systems Testing Automation

Pedro Santos Neto<sup>1,2</sup>, Rodolfo Resende<sup>1</sup>, and Clarindo Pádua<sup>1</sup>

<sup>1</sup> Computer Science Department,  
Universidade Federal de Minas Gerais (UFMG)  
{pasn, rodolfo, clarindo}@dcc.ufmg.br

<sup>2</sup> Computer Science and Statistic Department,  
Universidade Federal do Piauí (UFPI)  
pasn@ufpi.br

**Abstract.** This paper presents MODEST, a MethOD to hElp System Testing. MODEST can reduce the overall effort required during software construction, using an extended design specification produced in a UP-like software process. This specification is used to automate test generation and execution, decreasing the effort required during test activities. The method deals with Information Systems that follow an architecture composed of a user interface layer, a business rule layer and a storage mechanism abstracted by a persistence layer.

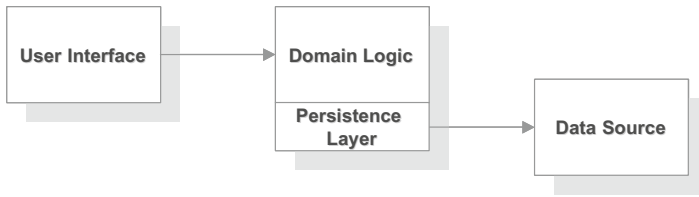
## 1 Introduction

Information Systems (IS) have a very important role in our lives. They are usually constructed using many components that operate together in order to retrieve, process, store, and distribute information. This information is generally used to help the analysis, control and decision making in an organization. Because of this, Information Systems became a pervasive technology, i.e., they are more noticeable by their absence than their presence.

Considering the relevance of Information Systems, a flaw during their operation can cause big damages. According to a report produced by NIST [1], the USA annual costs, due to an inadequate infrastructure for software testing, are estimated to range from \$22.2 to \$59.5 billion, only in 2002. Over half of these costs involve error avoidance and mitigation activities. The remaining costs involve additional testing resources that are consumed due to inadequate testing tools and methods. Many of these flaws are related to IS development.

These problems motivate our investigation on how to improve the efficiency of test activities, in particular the ones related to the following IS features:

- The ability to manage large amounts of persistent data.
- The management of a large number of users accessing information.
- The utilization of graphical user interfaces.
- The integration with other enterprise applications.



**Fig. 1.** The architecture supported by MODEST

In this work we present the **MethODod to hElp System Testing (MODEST)**. The main purpose of this method is the automatic generation and execution of system testing. This method takes advantage of the artifacts created during the development of an application. Most of the concepts in the method are process and modeling language independent, nevertheless, for the sake of readability and pragmatism, we describe MODEST in the context of the Unified Process - UP [2] guidelines. We use the UML [3] as prescribed by UP.

Our current work deals with softwares composed by a presentation layer, a business rule layer, and a storage mechanism abstracted by a persistence layer, as informally shown in Figure 1. This organizational structure is captured within the architecture of many Information Systems.

The remainder of the paper is organized as follows. Section 2 presents the method, detailing each activity. Section 3 briefly describes a MODEST compliant tool. Section 4 discusses an experimental study evaluating the method. Section 5 describes some related works. Section 6 presents our conclusions and future works.

## 2 MODEST Overview

The main objective of MODEST is to reduce the effort required in the system testing activities. Most aspects of the method are process independent, despite the use of several design artifacts produced during the system construction. Processes based on the UP process frequently create the artifacts used by MODEST. They are created using UML and it is simple to introduce the formalism required by the method.

Since we are interested in system testing, we use internal and external specification. We consider external specification the use case view of the system, showing the interactions among the system and the outside actors, without revealing the system internal structures. The internal specification corresponds to the logical view of the system and shows how to implement the system behavior in an effectively computable way [3].

Figure 2 shows the method activities. Activities A1 to A6 are start up activities. These activities detail how the software under test (SUT) should work. The method uses the information gathered in these activities to generate and execute tests, verifying if the software follows the specification. We assume that

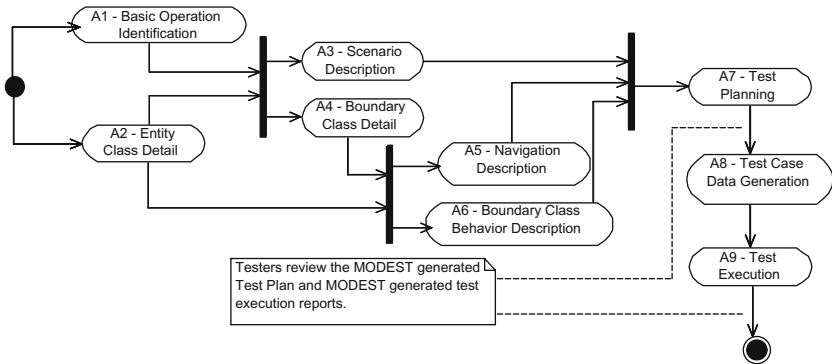


Fig. 2. Activity diagram showing MODEST activities

the specification serving as input to MODEST is correct, however the method describes several consistency checks as will be seen later. The correction of the specification can be assured by reviews, inspections or other techniques.

Activities A7 to A9 generate test procedures and test cases, execute the tests and create reports to be analyzed by the testers.

In the following subsections we briefly discuss each activity. We use a user authentication protocol as a running example. This simple example is composed by two use cases: *Login* and *User Management*. There are three windows in this example: *MainWindow*, used to authenticate users, *UsersWindow*, used to manage users (create, read, update, delete), and *SearchWindow*, used to search an element, in this case *Users*, in a collection.

In some sentences of this work, written as "*MODEST uses the information*", we are referring to MODEST as a tool and not a method description. We use this style since a method induces a corresponding tool, and in some places it is easier to explain how a MODEST compliant tool works, instead of describing the method prescriptions. We discuss a MODEST compliant tool in Section 3.

### 2.1 Basic Operation Identification

MODEST uses the persistence mechanism to generate and verify the test results. These operations are related to one of the CRUD (**C**reate, **R**ead, **U**ppdate, **D**ele) operations usually provided by a persistence layer.

During test generation, it is necessary to know the CRUD operations to populate the storage mechanism with the data required in the test. During the test execution it is necessary to verify the result of each persistent operation used, thus, if the *create* operation is used, it is necessary to verify if the appropriate data was correctly stored.

There are many ways to identify the persistence operations in a system. In our work, we decided to adopt the use of diagrams describing the CRUD operations of the persistence mechanism. These diagrams follow some naming and formatting conventions, facilitating the retrieval of the required information.

This option improves the quality of the project documentation, detailing how the persistence mechanism works.

## 2.2 Entity Class Detail

The entity classes usually model persistent information. These classes are usually independent of applications and they are strong candidates for reuse. MODEST uses these classes and their associations for storage mechanism population.

MODEST prescribes the specification of the entity classes and their associations, together with properties related to the entity class attributes. These extra properties must detail if an attribute is a key field, its type, valid values, minimal and maximal value, and if the attribute initialization is mandatory. As mentioned in Subsection 2.4, it is possible to create a link between an entity class attribute and a user interface field, in order to allow the test case generation.

There are some non mandatory entity classes properties that can be used for the generation of non-functional tests. If the query frequency, update frequency, maximum cardinality and maximum time for queries, are given MODEST prescribes how to populate the storage mechanism, and how to generate tests checking the time spent during CRUD operations.

## 2.3 Scenario Description

Scenarios illustrates a sequence of actions related to a behavior. In the UML, they are conventionally described using interaction diagrams. A scenario provides details about how to implement behavior in an effectively computable way. MODEST uses these scenarios, in order to discover what persistent operations are used and to help the determination of scenario expected result.

MODEST prescribes some modeling conventions for scenario description. First, each activated user interface command must be represented in the diagram by an operation with the same name. Second, these operations must receive, as parameters, the data required by its functioning. Third, there must be a mechanism for the identification of persistence operations used in a scenario. Fourth, all the exceptions thrown in the scenario must be detailed.

Figure 3 presents a scenario that follows MODEST prescriptions. *Login scenario* illustrates one of the *Login use case* executions. The first operation in the scenario corresponds to a user interface command (message number 1 in the diagram). The data required to execute this operation are the *login* and *password*. It is simple to identify the persistence operation invoked: there is a call to *read* method (message 7). Besides, all the exceptions related to this scenario are detailed, including error messages and the thrown conditions (messages 4, 6, 8 and 9).

MODEST prescribes how to create test procedures based commands activated in the scenario, the required data, and the executed persistence operations.

## 2.4 Boundary Class Detail

MODEST relies on boundary stereotyped classes representing user interfaces. These classes are central points for data input and output.

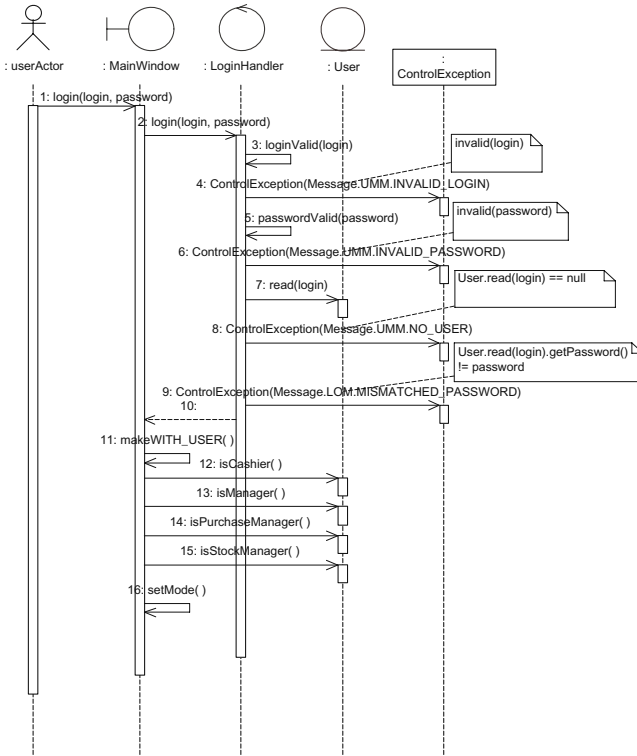


Fig. 3. Login scenario

MODEST prescribes the description of fields and operations visible by the end users. For instance, we modeled user interface fields in our example, using the stereotype <<field>> on attributes, and using the stereotype <<command>> on operations. MODEST recognizes this information and extracts the related data for use.

MODEST uses the specification of several relevant characteristics associated to fields and commands in order to generate and execute test cases. A field attribute must identify the style used in the target language (*TextField*, *PasswordField*, *CheckBox*, etc.), together with the source and the destination target of the field data. For instance, a source of a field can be an entity attribute, a software/hardware interface, a pre-defined value, e.g. system date, or a calculated value, e.g. the sum of two entity attributes. Fields related to an entity attribute must specify this association. This allows the automatic data generation, using the entity attributes properties.

A user interface command must also identify the style used in the target language (like *Button* or a *MenuItem*), the preconditions required for command activation, and postconditions specifying the system state after finishing the



command execution. MODEST does not prescribe a specific language to create these conditions. Any language with suitable expressive power can be used.

In our example, we used a simple language based on the persistence layer operations. Currently, we do not use OCL [4], in order to simplify our investigations. We are planning to adopt OCL in future versions our work. Below, we present a simplified version of the grammar for our language. This grammar is used to specify all the conditions presented in our examples.

```

CONDITION  EXP COMP EXP
COMP       = | = | == | != | =
EXP        FIELD | ENTITY | NUMBER | STRING | NULL | EXP OP EXP
OP         + | - | /
FIELD      List of user interface fields
ENTITY     ENT_ID.read(EXP) | ENT_ID.read(EXP).GET_METHODS
ENT_ID     List of system entities
GET_METHODS getATTRIBUTE()
ATTRIBUTE  List of entity attributes, starting with a capital letter
    
```

### 2.5 Navigation Description

MODEST prescribes the use of information related to user interface control transfer. This information can be represented in UML using a collaboration diagram involving the boundary class instances or by a class diagram with control transfer modeled by associations stereotyped as `<<link>>`.

Figure 4 shows control transfers of part of our example. Associations between boundary classes should be stereotyped with `<<link>>` whenever the association represent a possible navigation. Constraints on these associations are shown inside curly braces together with the corresponding commands. The control transfer from *UsersWindow* to *SearchWindow* objects occurs only when the *search* button is clicked and the *login* field is void (*login == ""*). MODEST specifies that all the constraints presented in this description must be exercised, i.e., used to create a test case.

### 2.6 Boundary Class Behavior Description

The expected behavior of the IS is directly related to the interaction of users with their user interfaces. We assume that in a typical design the exceptions are always associated with user interface error messages. We also assume that exceptions are modeled using the presentation of error messages conditions.

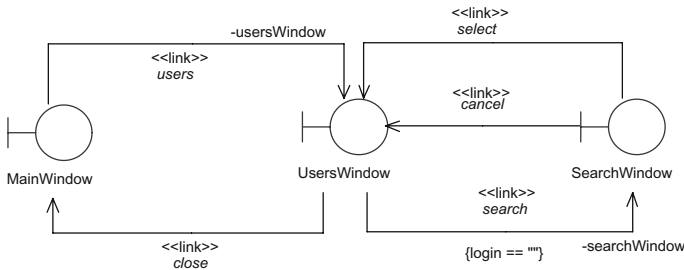
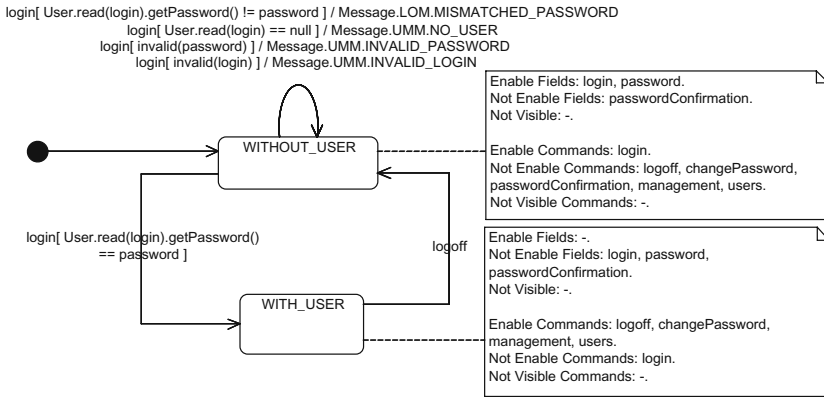


Fig. 4. A diagram showing the possible control transfers among windows



**Fig. 5.** *MainWindow* statechart excerpt

MODEST heavily uses the descriptions of the boundary classes behavior. Using this information it is possible to determine what needs to be tested and how to test it. However, it is necessary to provide these descriptions in a suitable format.

Statechart diagrams can be used to represent the boundary class behavior. Each boundary class needs to have a diagram showing the normal behavior and exceptions. The exceptions are modeled as guard conditions. Figure 5 shows an example of these descriptions for the *MainWindow* of our example. The user uses this window to login into the system. In order to have this user interface testable by using the method, we need to describe all the possible exceptions and user interface presentations.

The initial state of *MainWindow* is **WITHOUT\_USER**. This state describes one user interface presentation, indicating the enabled, disabled and invisible fields and commands. The user interface presentation needs to be tested after each command execution. All the exceptions related to **WITHOUT\_USER** are modeled in the statechart of the Figure 5. For instance, when the user clicks on the *login* button and the informed password does not match with the stored password, a message describing this is showed (Messages.LOM.MISMATCHED\_PASSWORD). This is represented by the first self-transition for the **WITHOUT\_USER** in Figure 5.

MODEST prescribes many consistency checks of the specification before test generation and execution. For instance, the *login* command must be enabled in the **WITHOUT USER** state, described in Figure 5, since there are many transitions related to this command. Besides, since there are some exceptional conditions also related to this command, these conditions must be described in every scenario that uses it, beyond the statecharts. There are many other consistency rules prescribed by MODEST, but we do not describe them here in order to save space.

## 2.7 Test Planning

In this activity, MODEST defines the test cases, indicating the associated test procedures and the conditions to be exercised. The test case data generation is done in the next activity.

MODEST will deal with every state and transition in the statecharts, and all the exceptions and scenarios.

Algorithm 1 depicts, in a simplified way, how MODEST defines the test case sequences. The general idea is always to search for any unused specification, in order to exercise it.

---

### Algorithm 1 Test Plan and Test Case Data Generator

---

<pre> Proc systemTestGenerator   for each useCase SUT use case do     for each currentWindow window related to       the useCase do         mark the currentWindow states as non exercised         mark the transitions in the statecharts of the         currentWindow as non exercised         startState the start state of the currentWindow         testCaseGenerator(startState)         for each navigation from the currentWindow do           navigationTestCaseGenerator             (navigationCondition, nextWindow)         end for       end for     end for   end for </pre>	<pre> Proc testCaseGenerator(currentState)   if currentState has been marked as exercised then     return   end if   for each currentTransition one of the possible   transition from currentState do     activatedCommand activated command in     the current transition     testProcedure test procedure related to activatedCommand     testDataGenerator(currentTransition, testProcedure)     mark the currentTransition as exercised     if nextState of the currentTransition = sourceState of     currentTransition then       testCaseGenerator(nextState)     end if   end for end for mark the currentState as exercised </pre>
<pre> Proc testDataGenerator(transition, testProcedure)   // Creates a test case from the analysis   // of the transition, including the condition   // to be exercised, defining data for input   // and storage mechanism. </pre>	<pre> Proc navigationTestCaseGenerator(condition, nextWindow)   // Create a test case from the analysis   // of the navigational condition to be exercised,   // defining data for input and storage   // mechanism. </pre>

---

## 2.8 Test Case Data Generation

After the determination of the test case sequences, it is necessary to specify the test input and output data. This is the main goal of this activity.

The test case inputs are determined analyzing the transition to be exercised, together with the properties of the involved fields. For instance, after the generation of test case sequences for our example, it is generated a test case related to the test procedure *Login*, based on the *Login* scenario, to exercise the first self-transition of Figure 5. In this transition there is a condition "*User.read(login).getPassword() != password*". This condition was created using our simple language, defined previously. MODEST prescribes the analysis of this condition, in order to generate input and persistent data.

In this example, it is necessary to generate a *User* with a valid *login*, in the storage mechanism, but with a *password* different from the stored *password*. MODEST prescribes the generation of valid values, using the properties specified in the user interface fields. As mentioned before, MODEST prescribe the specification of properties related to a user interface <<*field*>> indicating its source and target. The *login* field of our example is related to the *login* attribute

of *User* entity. This field is a *key field*, with *minimal size* two, *maximal size* eight, composed only by letters. So, it is simple to generate a valid *login* and *password*. MODEST also prescribes the determination of the persistent data required to run the test cases. In this example, it is required the existence of a specific *User*. The data fields required to this test case execution must be determined in this activity. An example is the creation of a *User* with a *login* "aaaa", *password* "aaaa". The test case data could be *login* "aaaa" and *password* "bbbb". This would assure the use of the desired condition (`"User.read(login).getPassword() != password"`).

At the end of this activity, several definitions are available: the input and persistent data required to run the test case, the conditions to be evaluated before and after each command activation, and the expected reached state after each command activation. Instead of specifying the determination of the output data, MODEST specifies how to express the conditions that must be satisfied after a command execution. This option has a low cost with reasonable benefits, compared to output data specification. These conditions are defined in the boundary class detail activity, and they are related to command pre and postconditions, together with the persistence operations used in the scenario related to the test procedure. These characteristics facilitate the determination of a test verdict for the test case, as we can see in the next activity.

## 2.9 Test Execution

MODEST prescribes the automatic test case execution. Many technologies can be used for this, such as computational reflection [5] and aspects [6].

Using reflection, for instance, it is possible to discover all the class attributes and operations during runtime, without requiring code changes. This information, combined with the boundary class description, can be used by a tool that obeys MODEST to put test case data in the appropriated fields and to execute the commands required in the test procedures.

In order to allow automatic test case execution, MODEST prescribes some conventions to be used in attributes and operations names. This is required in order to facilitate the identification of user interface fields and commands.

Executing a test case requires the storage mechanism population for the test, boundary classes loading, instantiation of user interfaces, setting of the appropriate data into the fields, and the execution of the commands in the corresponding test procedures, always checking the pre and postconditions, the reached state and the expected result related to the persistence operations.

MODEST prescribes the determination of a test verdict at the end of the test case execution. This verdict follows the UML Testing Profile specification [7]. There are four valid values: *fail*, *inconclusive*, *pass*, and *error*. A *pass* verdict indicates that the test case is successful and that the SUT has behaved as expected. A *fail* verdict on the other hand shows that the SUT is not behaving according to the specification. An *inconclusive* verdict means that the test execution cannot determine whether the SUT performs well or not. An *error* verdict tells that the test system itself and not the SUT failed.

### 3 A MODEST Compliant Tool

We developed a prototype to function as a MODEST compliant tool, named *MODESToo*. We show the *MODESToo* components in Figure 6. *MODESToo* reads an XMI [8] specification to extract the data related to software behavior. XMI is a standard format for metadata interchange for UML Tools. This format allows the interoperation of UML tools, provided they are able to generate an XMI file containing the software specification.

The *Extractor* is a *MODESToo* component responsible for data extraction. During this task, a consistency check is done, and the test procedures are generated based on the scenario descriptions.

The *Test Planner* generates the test case sequences to test the system. This component creates the test case sequences, detailing the test procedure to be used, the reached states during the test procedure execution, the conditions to be exercised, and the pre and postconditions to be verified.

The *Test Case Data Generator* is responsible for input data generation, based on the condition to be tested. It is also responsible to determine the persistent data required during the test case execution. The current *Test Case Data Generator* is based on our language, defined in the Subsection 2.3.

The *Populator* is the component responsible for storage mechanism population, based on the requirements of the test case. This can be a complex task, since an entity can have many associations and many mandatory fields not specified by the *Test Case Data Generator*. The *Test Case Data Generator* specifies the fundamental data for the test. For instance, if the *Test Case Data Generator* specifies the creation of a *User* with password "123456", the *Populator* must generate a *User* with a *login*, a *password*, a *name*, and belonging to some *groups*, since these properties are mandatory for a *User* creation. The password is the fundamental data for this example, and the other values are arbitrary.

The *Executor* is responsible for the system load, data input, and result analysis. It calls the *Populator* before each test case execution, to start a transaction and to create the persistent data required for test execution. This transaction is terminated with a roll back, at the end of the test, in order to remove all the

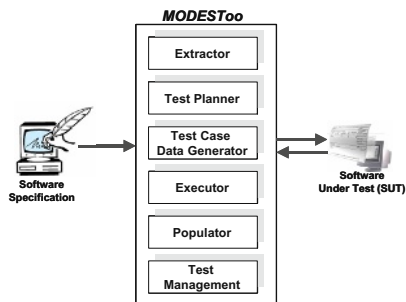


Fig. 6. A MODEST compliant tool

operations effects. This is the only component dependent on technology: if the SUT is a *Web based Information System*, it is required a *Web Executor*, if the SUT is a *Java-GUI based Information System*, it is required a *Java Executor*. All the conditions related to the test case must be evaluated by this component. The *Executor* is responsible for the test case verdict generation. It also stores the output test case data for future visualizations.

The *Test Management* is a graphical user interface for *MODESToo* administration. This component is used to start the method and to visualize the generated tests together with their execution. Besides, there are some facilities for test cases creation, decreasing the effort required in this task.

## 4 Experimental Study

We developed a study with the *motivation* to assess and better understand the use of MODEST. The *purpose* of the study was to evaluate, from the *perspective* of developers, the use of a *software process* instantiated *with* and *without* MODEST. We refer to these processes here as *MEP - MODEST Enhanced Process*, and *NEP - Non-Enhanced Process*. MEP and NEP processes are based on the UP software process. We adopted the recommendations of Basili, Selby, and Hutchens [9] in this study.

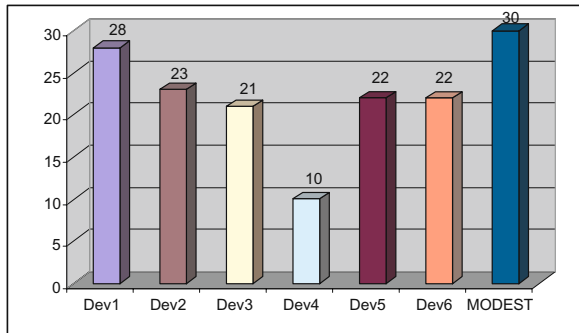
We started the experiment with sixteen software engineering students, along the experiment ten students dropped out, and only six of them finished the activities. We believe that the free character of the voluntarism can explain this. Three volunteers used MEP and three others used NEP to develop the Login use case discussed before. Table 1 shows the effort required by the two groups of students. The design specification was created using Rational Rose [10] modeling tool. Volunteers using MEP dedicated lower effort than NEP developers, since MODESToo automatically generates a partial test plan, including the test procedures and the test cases, based on the design specifications. Developers using NEP had an extra work to generate a test plan and to implement the test cases using Abbot [11] tool. In the experiment we considered design and test efforts.

Both groups had a four-hour explanation on the techniques involved in the experiment. The MEP group had an extra four-hour class detailing the MODEST conventions required in the design. The artifacts generated by the developers were reviewed by the paper authors.

Our experiment was designed with the goal of answering the following main questions:

**Table 1.** Effort spent in the experiments (in hours)

MODEST			NORMAL		
Dev1	Dev2	Dev3	Dev4	Dev5	Dev6
3,50	5,32	3,72	9,75	18,47	20,31



**Fig. 7.** Number of failures detected per developer during experimental study

- What is the extra time required to create a software design model extended with MODEST prescriptions?
- Is it possible to decrease the construction effort using MODEST?
- What is the quality of the MODEST generated tests?

The experiment analyzed the following two hypotheses: (i) MEP design specification creation effort is the same as in NEP, and (ii) MEP overall effort is the same as in NEP.

*Hypothesis (i)* was confirmed, and *hypothesis (ii)* was considered false, from the analysis of the collected data (95% confidence interval). In other words, the extra-time required by MODEST extensions was compensated by the time gained in test activities, and the overall effort reduction was significant.

Additionally, we asked all the volunteers, MEP and NEP, to build a test plan and to implement this test plan, in order to evaluate the quality of MODEST generated tests, even though MEP volunteers were not supposed to deal directly with tests. From past software engineering classes student projects we retrieved the thirty most common failures. We injected a fault for each of these failures in the experiment SUT. Figure 7 shows the number of detected failures per developer, using the manually generated tests, and MODEST generated tests. We concluded, from this figure that there are failures detected by MODEST and not detected by the volunteers, therefore the failures detected by MODEST are not trivial. Volunteers detected an average of 21 faults, with a standard deviation of 5.93. Each injected fault was detected by at least one of the manually generated tests, showing that injected faults were not specially contrived.

## 5 Related Work

Offutt and Abdurazik [12] claim to be the first group to formalize a testing technique based on UML. They created a technique for test data generation based on statecharts. This technique was innovative but difficult to deal with systems containing many classes with many concurrent statecharts.

Briand and Labiche [13] developed a method for functional system testing named TOTEM (Testing Object-oriented systems with the unified Modeling language). This method derives test requirements from artifacts created during analysis. It requires the creation of an activity diagram showing the use case dependencies to allow the test generation. They do not describe completely how to generate the test cases, how to start the tests if some data is required, and how to reduce, effectively, the huge amount of generated tests. They do not report a TOTEM compliant tool and method evaluation.

The AGEDIS [14] project created a methodology and tools for automated model driven test generation and execution for distributed systems. The project includes an integrated environment for modeling, test generation, test execution, and other test related activities. Their own work point out some problems. These problems are related to the modeling language conventions, the use of statechart as the main behavioral description of the SUT, and the language used as action language. We believe that another problem is the use of some uncommon artifacts for test generation, like the test generation directives.

In the Siemens Research Center was developed an integration testing technique for components using UML statecharts [15]. This method requires the definition of the dynamic functioning of the components using statechart diagrams adopting CSP (*Communicating Sequential Processes*) for describing the communication between two components. They do not report how their technique blends with a software process.

IRISA developed a method and a tool for automatic generation of test cases from UML design specification [16]. The UML model describing the system must have a class diagram showing the class associations, a object diagram showing the start configuration and a statechart for each class. The test objectives must be specified in sequence diagrams with some restrictions that can make their creation hard. Their work lacks some directives, for example, there is no comprehensive description about how the start configuration is set up during the test executions.

## 6 Conclusion

This paper presents MODEST, a **MethODology to hElp System Testing**. MODEST can reduce the overall effort required during software construction, using an extended design specification usually produced by software processes that follow the UP guidelines. This specification is used to automate test generation and execution, decreasing the effort required during test activities. The method deals with Information Systems composed by a presentation layer, a business rule layer, and a storage mechanism abstracted by a persistence layer, as informally shown in Figure 1.

Adapting MODEST for use in a UP-like software process was simple. We extended some activities and the registered impact was not significant, as mentioned in *hypothesis (i)* of Section 4. Besides, the overall effort reduction was significant, since MODEST automates the majority of testing activities. We can



conclude that the failures detected by MODEST are not trivial, since the set of volunteers did not detect all of them. Additionally, each injected fault generated a failure detected by at least one of the individual volunteers, showing that the injected faults were not specially contrived.

During our experiments we noticed that the number of generated test cases was comparable to the number of manually generated ones. We are planning to enlarge the size and the complexity of the systems in our experimental study, in order to evaluate MODEST scalability.

Currently, we are working in two different projects related to MODEST. We are developing a MODEST Mutation [17] Tool to help the evaluation of MODEST generated tests, and we are extending MODEST to deal with more sophisticated architectures.

## References

1. NIST, Planning Report 02-3, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, 2002, <http://www.nist.gov/director/prog-ofc/report02-3.pdf>, last access on November 2004.
2. Jacobson, I., Rumbaugh, J., and Booch, G., *The Unified Software Development Process*, Addison Wesley, 1999.
3. Rumbaugh, J., Jacobson, I., and Booch, G., *The Unified Modeling Language Reference Manual*, Addison Wesley, 1999.
4. Warmer, J., and Kleppe, A., *The Object Constraint Language*, Addison-Wesley, 2nd edition, 2003.
5. Maes, P., Concepts and Experiments in Computational Reflection. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'87)*, pages 147-155, Orlando, Florida, December 1987.
6. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., and Irwin, J., Aspect-Oriented Programming, *In European Conference on Object-Oriented Programming (ECOOP'97)*, LNCS 1241, pages 220-242, Finland, June 1997.
7. OMG, *UML Testing Profile*, Object Management Group document, March 2003, available from <http://www.omg.org>, last access on November 2004.
8. OMG, XML Metadata Interchange (XMI). *Object Management Group document*, July 1998, available from <http://www.omg.org/>, last access on November 2004.
9. Basili, V., Selby, R., Hutchens, D., Experimentation in Software Engineering, *IEEE Transactions on Software Engineering*, volume 12, number 7, July 1986.
10. Rational Software Corporation, *Rational Rose User's Guide*, available from <http://www.rational.com/>, last access on November 2004.
11. Abbott, *Abbot Java GUI Test Framework*, available from <http://abbot.sourceforge.net/>, last access on November 2004.
12. Offutt, J., and Abdurazik, A., Generating Tests from UML Specifications, *Proceedings of the 2nd Unified Modeling Language Conference (UML'99)*, pages 416-429, Fort Collins, CO, USA, October 1999.
13. Briand, L., and Labiche, Y., A UML-Based Approach to System Testing, *Proceedings of the 4th Unified Modeling Language Conference (UML'01)*, pages 194-208, Toronto, Canada, October 2001.

14. Hartmann, A., and Nagin, K., The AGEDIS Tools for Model Based Testing, *International Symposium on Software Testing and Analysis (ISSTA 2004)*, Boston, Massachusetts, USA, July 2004.
15. Hartmann, J., Imoberdorf, C, and Meisinger, M., UML-Based Integration Testing, *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2000)*, pages 60-70, Portland, Oregon, United States, August 2000.
16. Pickin, S., Jard, C., Le Traon, Y., Jéron, T., Jézéquel, J.-M., and Le Guennec, A., System Test Synthesis from UML Models of Distributed Software, *Proceedings of 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2002)*, pages 97-113, Houston, Texas, November 2002.
17. DeMillo, R., Lipton, R., and Sayward, F., Hints on Test Data Selection: Help for the Practicing Programmer, *IEEE Computer*, volume 11, pages 34-41, April 1978.

# Model-Based System Testing of Software Product Families<sup>1</sup>

Andreas Reuys, Erik Kamsties, Klaus Pohl, and Sacha Reis

University of Duisburg–Essen, Software Systems Engineering,  
Schuetzenbahn 70, 45117 Essen, Germany

{Reuys, Kamsties, Pohl, Reis}@sse.uni-essen.de

**Abstract.** In software product family engineering reusable artifacts are produced during domain engineering and applications are built from these artifacts during application engineering. Modeling variability of current and future applications is the key for enabling reuse. The proactive reuse leads to a reduction in development costs and a shorter time to market. Up to now, these benefits have been realized for the constructive development phases, but not for testing. This paper presents the ScenTED technique (Scenario based TEst case Derivation), which aims at reducing effort in product family testing. ScenTED is a model-based, reuse-oriented technique for test case derivation in the system test of software product families. Reuse of test cases is ensured by preserving variability during test case derivation. Thus, concepts known from model-based testing in single system engineering, e.g., coverage metrics, must be adapted. Experiences with our technique gained from an industrial case study are discussed and prototypical tool support is illustrated.

## 1 Introduction

Software product family engineering (PFE) is an emerging discipline. The goals of PFE are to reduce development costs and Time-to-Market as well as to increase quality of individual applications [4]. An essential concept is proactive reuse [14]. Following this concept, PFE is structured into domain engineering (*development for reuse*) and application engineering (*development with reuse*) [21]. Reusable artifacts are created during domain engineering and are reused during application engineering to create customer-specific applications.

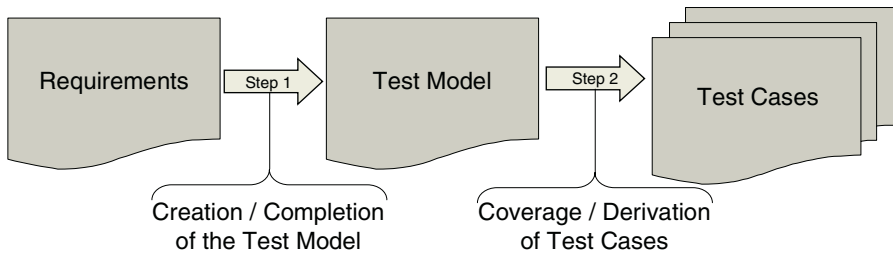
The increased productivity in product family engineering requires a more efficient test approach than those used in single system engineering. Testing consumes up to 50% of the total effort in single system engineering [1]. This percentage increases in PFE, because the effort for constructing applications decreases due to comprehensive reuse. Testing becomes a serious bottleneck of product family development.

---

<sup>1</sup> This work was partially funded by the CAFÉ-project “From Concept to Application in System Family Engineering“ (BMBF, Foerderkennzeichen 01 IS 002 C) and the ITEA Project ip02009 FAMILIES ”FACT-based Maturity through Institutionalisation Lessons-learned and Involved Exploration of System-family engineering“, Eureka Σ! 2023 Programme.

We argue that the idea of proactive reuse should be extended to product family testing to accelerate this activity [12]. Following this idea, reusable test cases are created in domain engineering, which are reused for testing an application.

The ScnTED technique (Scenario based TEst case Derivation) adapts model-based testing to product family engineering and supports the proactive reuse of test cases. Model-based testing is an approach to systematically derive test cases in single system engineering. In essence, model-based testing consists of two steps (see Fig. 1). A test model is built from the requirements. It is common to use state charts [17] or activity diagrams [9] as representation for the models. In a second step, test cases are created using coverage criteria or other derivation techniques.



**Fig. 1.** Model-based Testing in Single System Development (adapted from [6])

Model-based testing offers several advantages, e.g. test cases can be created in a systematic, i.e., repeatable fashion, and stopping rules can be defined [17]. Therefore, model-based testing is considered a prerequisite for automated test generation [3][6]. Another very important aspect is that test engineers validate the requirements by creating the test model. Defects in requirements, such as ambiguities and incompleteness, may be detected during the development of the test model, which is cheaper than correcting them in later development phases. These benefits can be realized for product family engineering by adapting model-based testing.

Variability is the key challenge for adapting model-based testing in product family engineering. The variability of a product family specifies the differences among applications to be build and is defined during domain engineering. Functionality is called a variant whenever it is not planned to be part of all applications. For example *pay per credit card* is a variant in an eShop, because it is not part of all applications. To cope with variability, it is necessary to adapt the test model, its creation process, test case derivation, and representation of test cases. In summary, the goals of our approach are:

- to achieve a reduction in test case development effort compared to the use of single system techniques in product family engineering, and
- to realize the benefits of model-based testing for product families by considering variability in test models and techniques.

In the following, the related work on product family testing is reviewed.

## 1.1 Related Work

Three approaches support the idea of extending proactive reuse to product family testing. McGregor [13] and Geppert et al. [17] create reusable test cases during domain engineering, but these approaches are not model-based. The test cases are derived from natural language requirements [13] and generalized from existing test cases [7].

Nebut et al. [16] follows also the idea of proactive reuse. They consider scenario fragments in domain engineering that are assembled to test case scenarios. However, there is no test model that guides the assembling of these fragments during domain engineering. Dependencies between use cases are specified in a use case transition graph, but test case scenarios are only derived for specific applications when the variability has already been bound.

Hartmann et al. [10] use an activity diagram as test model, which contains variability, but test cases are derived only in application engineering. Therefore, it is a model-based testing approach, but does not consider the reuse of test cases. Bertolino and Gnesi [2] do not use a test model, but a structured test specification that contains variability. Test cases are created for each application based on this specification.

In summary, the approaches by McGregor, Geppert et al. and Nebut support the idea of extending proactive reuse to product family testing. Hartmann et al. support the idea of model-based testing in product family engineering. However, there is no approach for product family testing up to now, which combines pro-active reuse with the benefits of model-based testing.

## 1.2 Overview

In this paper the ScnTED technique (Scenario based TEst case Derivation) is presented. ScnTED is a model-based technique for system testing in product family engineering. The key idea of the technique is to create reusable test case scenarios in domain engineering and to reuse these test case scenarios in application engineering.

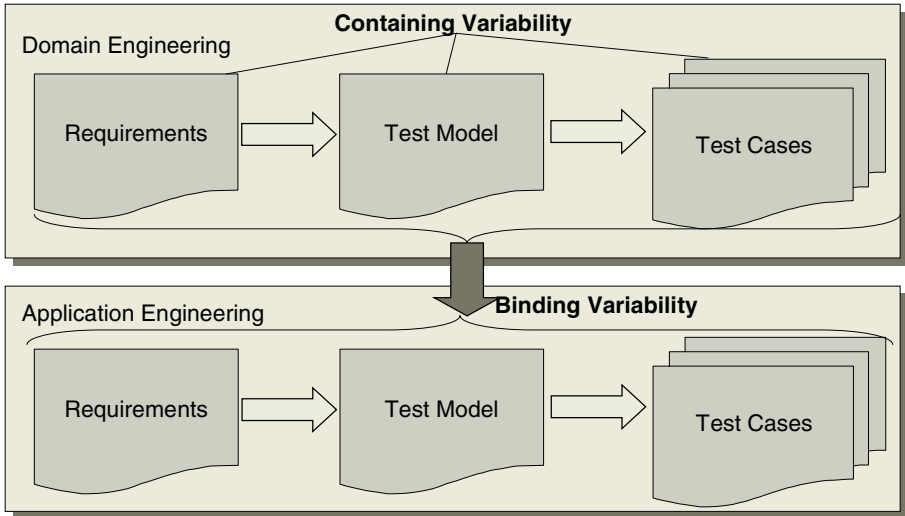
The adaptation of model-based testing for product family engineering and its realization in ScnTED is explained in Section 2. The ScnTED activities for model-based testing are described in Section 3. ScnTED has been used in an industrial setting where a case study has been performed to measure the reuse benefit. This case study is described in Section 4. Section 5 gives an overview on prototypical tool support for ScnTED. The paper concludes with a summary and outlook on future work.

## 2 Model-Based System Testing in Product Family Engineering

In this section we present an adaptation of model-based testing. Test models used within the ScnTED technique are defined, whereas the technique itself is explained in the next chapter.

Model-based testing has to be performed in domain engineering as well as in application engineering. It has to be conducted in domain engineering for two reasons. First, the domain test model and the test cases are used to facilitate an early validation of the domain requirements. Second, the test cases are created for reuse in application engineering.

Test models must contain variability and techniques must consider this variability in domain engineering (Fig. 2). Commonalities and variability are specified in domain requirements [8]. As requirements include variability, test cases must also contain explicit variability information. Test cases that include variability are called variant test cases. Test cases without variability are called common test cases and can be applied to all applications.



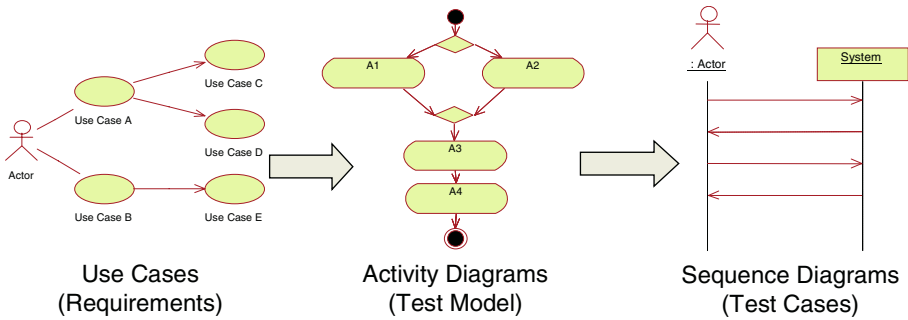
**Fig. 2.** Model-based Testing in Software Product Family Engineering

The model-based testing approach has to be conducted during application engineering, too. Thereby, no new test models must be created, but the test models that contain variability must be adapted to application specific needs. Application test models are required for two reasons. First, test models and test cases document the test activities. This is necessary when customers or laws and standards demand a proof that requirements have been tested successfully (e.g. U.S. Food and Drug Administration requires a proof of requirement coverage in testing). Second, an application test model is required to identify reusable test cases from domain engineering and to derive application-specific test cases.

In application engineering, the application engineer defines requirements with a customer. The application test model is built based upon the application requirements and by reusing the domain test model. Variability within the domain test model is removed and new requirements from customers may be added to get the application test model. Test cases are identified and derived in the following two steps. First, the reusable test cases from domain engineering are selected. Some of these test cases require an adaptation due to the selected variability in the application. Secondly, new test cases must be derived, if new requirements were incorporated into the test model.

The ScnTED technique is based on the assumption that requirements have been specified as use cases (Fig. 3). Activity diagrams are used as test model from which

test case scenarios are derived. The test case scenarios are specified in sequence diagrams. Test case scenarios describe the test engineer's actions and the responses of the system without specifying concrete test data.



**Fig. 3.** Models Used within ScenTED

We have chosen use cases as requirements specification, because use cases are well suited to elicit and document customer requirements for information systems in single system engineering as well as in product family engineering (see [8] for a survey on the extension of use cases for PFE). Use cases serve also as good starting point for system testing as they describe the system behavior from an external point of view, which is the focus of the system test. Activity diagrams or state charts are the most common test models to represent possible scenarios of use cases in single system engineering and have already been used for model-based system testing [9]. Sequence diagrams are simple forms of representation to negotiate the test scenarios with requirements engineers and can be incorporated into commercial test tools [11].

Test case scenarios are related to test case specifications. A test case specification refines a test case scenario and comprises detailed test inputs (e.g., 4711 as valid PIN number), expected results (e.g., system response *Enter withdraw amount*), additional information (e.g., to use a touch screen or number block), and test scripts relevant to this scenario. Test case scenarios can be generated automatically, but test case specifications are usually developed manually, which is a time-consuming task. Thus, it is desirable to create both test case scenarios and specifications in domain engineering and to reuse them during application engineering [4][7][13].

We do not describe the creation of the test case specification, as it is out of the scope of this paper. However, it is important to recognize that the reuse of test case scenarios implies the reuse of associated test case specifications and thus reduces the amount of test specifications that have to be created manually.

### 3 Product Family System Test with ScenTED

In this chapter, the activities of ScenTED for the model-based system testing are explained. During activity 1 (Fig. 4) requirements specified in use cases are modeled as domain activity diagram containing variability (Section 3.1). During activity 2 in

domain engineering test case scenarios are derived using an adapted coverage criterion for product family engineering (Section 3.2). Activity 3 comprises the adaptation of these test case scenarios for a customer-specific application (Section 3.3). Traceability information created during the first two activities is required to perform the third activity (depicted by the dotted arrows).

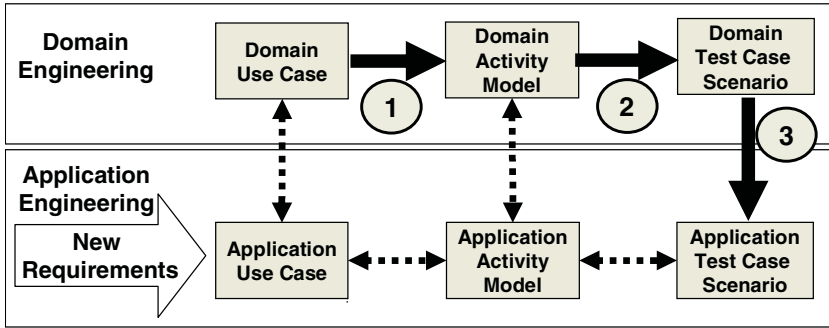


Fig. 4. ScenTED Activities

### 3.1 Creating the Hierarchical Activity Diagram

Within the first activity of ScenTED (Fig. 4) domain use cases are taken as input and a hierarchical activity diagram is created. In a first step, one activity diagram per use case is created that represents all scenarios specified within the use case. In a second step, the activity diagrams are integrated into an overall activity diagram enabling the derivation of end-to-end scenarios.

A prerequisite to use activity diagrams in domain engineering is the ability to represent variability in the control flow [20]. A variation point is documented as a special decision within the activity diagram. The variants are represented as different control flows, e.g. as sequences of activities at the variation point [19]. Variants may be optional, alternative, or co-existing. *Optional* means that the variants may be chosen additionally for an application ([0..1 out of 1]-dependency). *Alternative* is a [0..1 out of n]-dependency, meaning that one variant out of many variants may be chosen at maximum. *Co-existing* enables the selection of zero to m variants out of n possible variants ([0..m out of n]). A variation point is called *mandatory*, if at least one variant has to be chosen. In that case, an alternative-dependency has a cardinality of [1 out of n] and a co-existing-dependency is [1..m out of n]. Variation points that are not mandatory are also called *optional*.

For system testing it is necessary to create end-to-end scenarios. Therefore, the activity diagrams are arranged in one hierarchical diagram. The starting point to create the hierarchical diagram is the use case model. The top level of the hierarchy is a diagram, which contains the use cases that a user can perform directly. These use cases are modeled as activities, each activity includes another activity diagram. These activity diagrams on the second level model the behavior within the use cases. Included use cases are modeled on the next level and variants on the last level.



The creation of the hierarchical activity diagram is shown for the example of an e-Shop, depicted in Fig. 5. The use case diagram in Fig. 5a) shows that the buyer may *Register* or *Buy goods*. The use cases *Search goods* and *Search catalogue* are not directly initiated by the buyer, but included in use case *Buy goods*. Diagram b) shows the resulting overall activity diagram. The user may *register* or *buy goods* as he enters the eShop. These two activities are the only activities in the activity diagram as these are the only activities directly triggered by the actor.

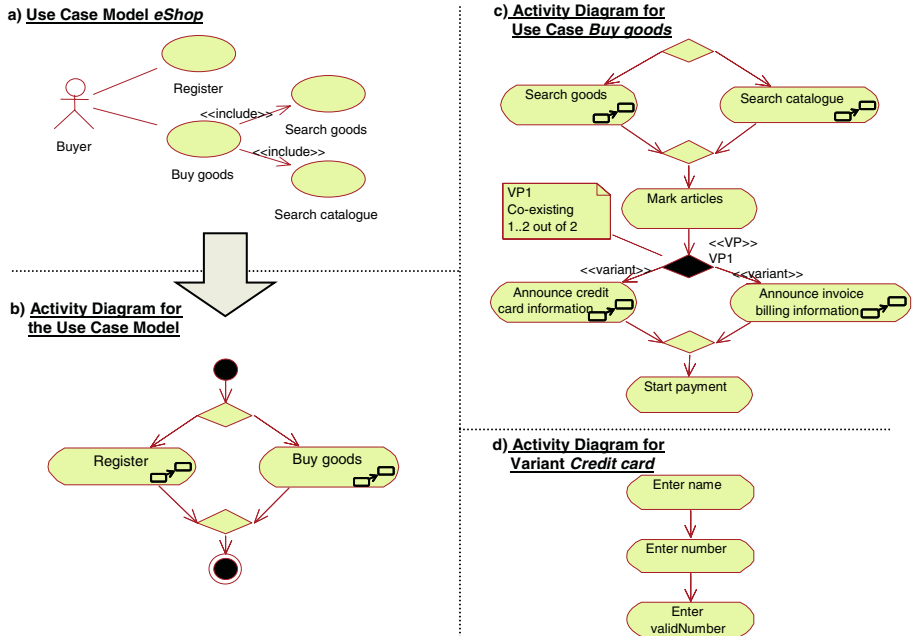


Fig. 5. Activity Diagrams containing Variability

The activity diagram for a single use case is shown in diagram c). This is the diagram for use case *Buy goods*, which is the refinement of the activity *Buy goods* in the activity diagram on the top level. The activity diagram for *Buy goods* represents all possible scenarios specified in the use case. Variability defined within the use cases is represented by the specific decision point (stereotype <<VP>>) and outgoing branches that are additionally marked with stereotypes [19]. Customers of future applications must choose if their application should provide the *credit card* variant, the *invoice billing* variant, or both variants as VP1 is specified as co-existing.

The next level of activity diagrams bears the included use cases *Search goods* and *Search catalogue* as well as diagrams for the variants *CreditCard* and *InvoiceBilling*. The activity diagram for variant *Credit Card* is depicted in diagram d).

### 3.2 Deriving Test Case Scenarios

In the second ScenTED activity, the test model created during domain engineering is used to derive test case scenarios. This is achieved by using a coverage criterion. The adaptation of well known code coverage criteria for model-based testing, esp. activity diagrams is also considered in [3]. Important criteria are the statement coverage criterion, the path coverage criterion, and the branch coverage criterion. The statement coverage criterion is only a weak criterion whereas the path coverage criterion leads to a huge amount or even an infinite number of scenarios in a complex system [15]. The branch coverage criterion is most commonly used in single system development. Therefore, we adapted this criterion for product family engineering within ScenTED.

The original branch coverage criterion is fulfilled when all branches of the activity diagram are covered by at least one scenario. But variability within the activity diagrams may lead to the fact that branch coverage is no more fulfilled during application engineering as a result of bound variants. Consider the following example of two scenarios for use case *Buy goods* (Fig. 5c), which fulfill the branch coverage criterion:

SC<sub>1</sub>: {Search goods, Mark articles, Announce credit card information, Start payment}  
 SC<sub>2</sub>: {Search catalogue, Mark articles, Announce billing information, Start payment}

If only variant *Credit Card* is selected for an application, then the second scenario is invalid. So the original branch coverage criterion fails when performed during domain engineering. Therefore, we adapted the branch coverage criterion:

*Each branch of the control flow for every possible application has to be covered by at least one scenario.*

In order to derive scenarios that fulfill this criterion, the key idea is to abstract temporarily from the variability, to derive the scenarios, and afterwards to detail the variability again. This works for all mandatory variation points, but only if the variability in the flow of events has only local impact. The application of this idea on the example leads to the following temporary scenarios in domain engineering for *Buy goods*.

SCV<sub>temp1</sub>: {Search goods, Mark articles, { }<sub>VP1</sub>, Start payment}  
 SCV<sub>temp2</sub>: {Search catalogue, Mark articles, { }<sub>VP1</sub>, Start payment}

These two temporary test case scenarios cover all branches except for the branches containing variability. Therefore, the variants have to be added now, thereby preserving variability. This leads to the following two domain test case scenarios.

SCV<sub>1</sub>: {Search goods, Mark articles, {Announce credit card information, Announce billing information}<sub>VP1</sub>, Start payment}  
 SCV<sub>2</sub>: {Search catalogue, Mark articles, {Announce credit card information, Announce billing information}<sub>VP1</sub>, Start payment}

This approach works for the example, but only because of the mandatory variation points and because the variation point is independent, i.e. the flows of events of both variants continue with the same activity (Start payment). For optional or dependent variation points this approach has to be modified. In a first iteration the variants and variation points are not considered during the coverage criterion. This leads to an

initial set of test scenarios that fulfills the branch coverage, besides the branches with variability. In a second iteration, the branches of the variants are covered and integrated into end-to-end scenarios. The resulting step is to merge the scenarios from the two iterations to the set of domain test case scenarios.

Test case scenarios contain more information than scenario steps. The result of each test case scenario step must be verifiable. Therefore, test data and the expected results of each step must be documented. Furthermore, the preconditions and post-conditions denoted in use case must be considered.

Diagram e) in Fig. 6 shows the domain test case scenario  $SCV_1$  for use case *Buy goods*. Sequence diagrams are used in ScenTED to represent the test case scenarios. Additional information on test data and expected results are documented in notes. This information cannot be derived completely by the model-based approach. The test data is often difficult to generate from the test model. Moreover, the expected result is mostly determined by the test engineer. This information can be described in the domain test case scenario and reused during application engineering to reduce effort.

The variability preserved in the activity diagrams is still included within test case scenarios. The test case scenario contains variation point  $VPI$  as well as the corresponding variants. The variable region within the diagram is emphasized with a note.

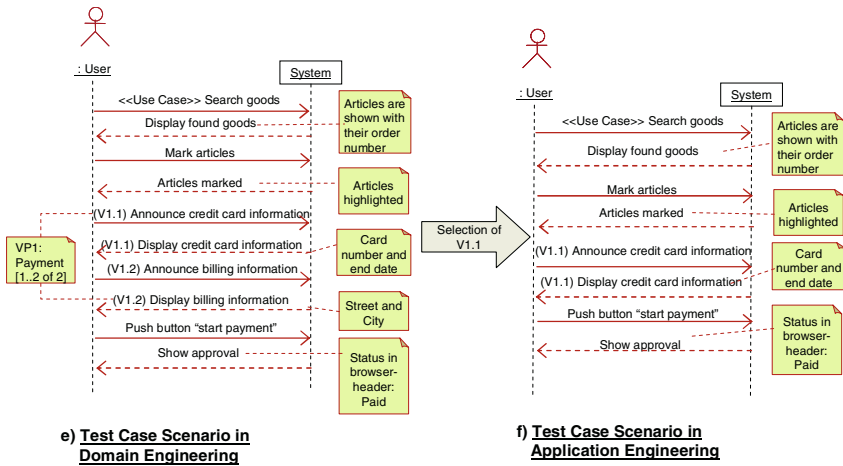


Fig. 6. Binding Variants in Test Case Scenarios

### 3.3 Deriving Test Case Scenarios for Specific Applications

The third ScenTED activity uses the domain test case scenarios and the application requirements (use cases) to create the set of application test case scenarios. The key idea is to reuse test case scenarios from domain engineering. The variability within the domain test case scenarios must be bound for the derived application and the test scenario is used to document the test case execution.

Customer requirements are elicited at the beginning of application engineering. Thereby, possibilities of the software product family are explained to the customer

and the customer defines the application requirements [8]. The result is documented in the application use case model.

Basically, the test engineer has to perform the same two steps as in domain engineering, but the test models created during domain engineering serve as blueprint. In the first step, the hierarchical activity diagram is used and adapted according to the application requirements. Essentially, variants are chosen for the specific application. Furthermore, it is possible that use cases or use case scenarios may be added or deleted. In the second step, the test engineer determines which test case scenarios derived during domain engineering are still valid for the application, i.e. all common test case scenarios are reviewed to decide if they are applicable or out-dated by changes. Thereby, the following cases may be distinguished:

- Test case scenarios for unchanged commonalities are directly reused.
- Test case scenarios for changed functionality are left out.
- Test case scenarios for variability are bound with the according variants.

Furthermore, additional, application-specific test case scenarios must be created, if there are still uncovered branches within the application activity diagram, esp. for changed functionality. The original branch coverage can be used as the application activity diagram does not contain variability anymore.

Fig. 6 shows the binding of variability for the test case scenario  $SCV_1$ . The variant  $V1.1$  *Pay per credit card* (diagram e)) is chosen for the application. The resulting scenario is shown in diagram f), which does exclude any not-selected variant.

The three activities of ScenTED can reduce effort in test case creation for software product families. This is achieved by reusing test case scenarios created during domain engineering. The time spent in the creation of test case specifications (e.g. scripts, data, and results) can be economized if the domain test case scenarios are related to the specifications and are reused in application engineering.

The described technique has been formalized and implemented into a prototype. The prototype will be sketched in Section 5.

## 4 Case Study: ScenTED at Siemens

ScenTED has been applied within a case study at Siemens AG Medical Solutions HS IM. The goal of the case study was to reduce the effort for test creation by reusing test case scenarios.

Siemens Medical Solutions develops workplaces for radiologists. The radiologists' tasks include the creation of an examination request, the filming of images, and the completion of a patient report. The considered product family supports the tasks of creation and administration of patient records and image data. The recorded data is stored on a central server. The creation of a report and image-post processing take place at a client workstation. Several different clients are developed based on the same documents (requirements, architecture, and code). The clients have different qualitative variants as well as functional variants. The qualitative variants define workstations, which either view images on a normal monitor or on several dedicated, high resolution monitors. The functional variants define applications that are only

able to view images or may copy the images to other embedded applications, e.g. 3D viewing applications.

#### 4.1 Adaptation and Application of ScenTED

This section describes the adaptation of the three ScenTED activities to Siemens-specific needs. Use of commercial tools within the application of ScenTED is shown.

*Creation of the Activity Diagram:* The developers at Siemens already use a hierarchical activity diagram. The described notation of variability in activity diagrams (see Section 3.1) has been introduced and adapted. Fig. 7 shows such an activity diagram. The notation was slightly changed. The stereotypes specify now the activities that can be used in the specific application (Product A/B/C). A stereotype has been defined for each meaningful combination of applications. The activity diagram therefore represents the domain test model with its possible variants as well as the application test model, and thereby the connection between variant functionality and applications is made explicit. Activities without stereotype are supported by all applications. Thus, these activities represent common functionality.

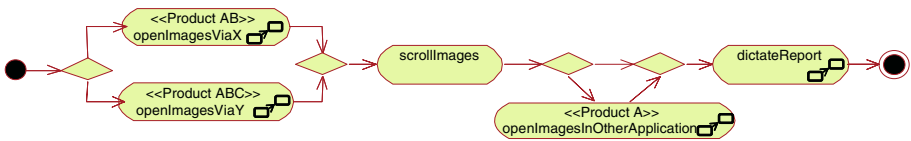


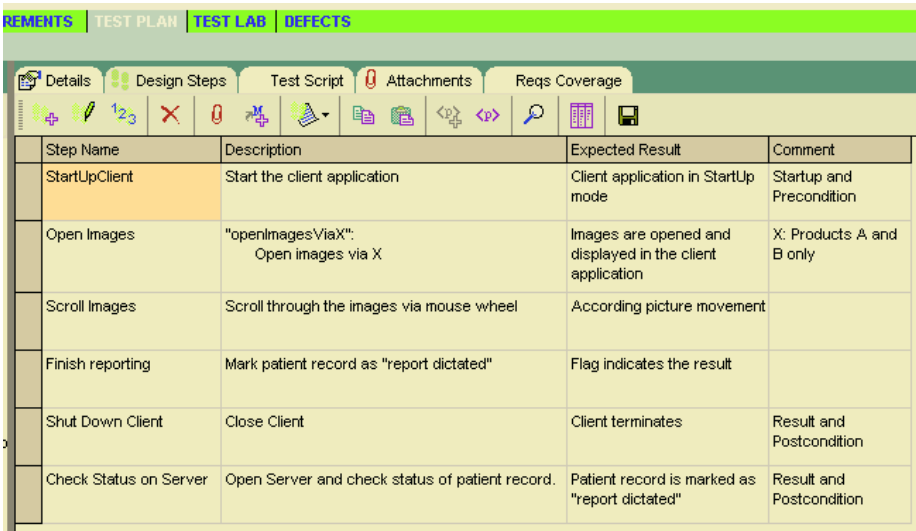
Fig. 7. Adapted Activity Diagram used at Siemens

*Derivation of Domain Test Case Scenarios:* Test case scenarios are derived from the activity diagram using the adapted coverage criterion. There were a lot of optional variation points, because one application realizes a lot of functionality, but others do not. According to the described approach in Section 0, the branches without variability are considered first, creating the test case scenarios for common functionality. Then test case scenarios are derived that contain variability. The test case scenarios are supplemented with information on application restrictions, i.e. on which application they are usable. This information is taken from the stereotype in the activity diagram. The resulting scenarios are also modeled in IBM Rational Rose. The diagrams include the test data and expected results.

*Derivation of Test Case Scenarios for an Application:* Test case scenarios for an application are derived from domain test case scenarios. All common test case scenarios are reused and test case scenarios that are possible for the considered application (specified within the domain test case scenarios) are selected additionally. Application test case scenarios are administrated in the test tool Mercury TestDirector (Fig. 8). The steps to ensure the precondition are marked as *Startup and Precondition* in the comment field. The field *Expected Result* describes the result for each step. The steps to validate the overall test case result and post-condition are specifically marked in the comment. Furthermore, the comment contains the specification in which application the test case scenario is applicable.

The benefit from the test tool is that test cases for common behavior and test steps from variability containing test cases may be reused in a copy-and-paste manner with only minor adaptations. This is working for the test case design (see Fig. 8) as well as for the underlying test script.

An idea developed from the Siemens test engineers is to define a library that contains a test script fragment for each activity within the activity diagram. Each fragment can be called with a parameter for test data. This library thus includes test fragments for all activities within the Rational Rose model. New test case scenarios can be created using drag-and-drop whenever the scenario is derived from an existing and already covered activity diagram.



Step Name	Description	Expected Result	Comment
StartUpClient	Start the client application	Client application in StartUp mode	Startup and Precondition
Open Images	"openImagesViaX": Open images via X	Images are opened and displayed in the client application	X: Products A and B only
Scroll Images	Scroll through the images via mouse wheel	According picture movement	
Finish reporting	Mark patient record as "report dictated"	Flag indicates the result	
Shut Down Client	Close Client	Client terminates	Result and Postcondition
Check Status on Server	Open Server and check status of patient record.	Patient record is marked as "report dictated"	Result and Postcondition

Fig. 8. Test Case Scenario in Mercury TestDirector

## 4.2 Results

ScenTED has been evaluated regarding two aspects: the amount of reused test cases and the test engineer's opinion whether ScenTED supports reuse of test cases. Therefore, the number of reused test case scenarios was determined at the end of the testing phase and a questionnaire was given to the test engineers.

The analysis of the applied test case scenarios led to the following results: 27 end-to-end test case scenarios were created during domain engineering. 63 test case scenarios were derived from these domain scenarios during application engineering. Each of the 27 test case scenarios had to be implemented in the TestDirector first. The 63 test case scenarios were reused from the originating 27 scenarios. Applying single system testing techniques would result in the implementation of 63 independent test case scenarios. Therefore, the reuse-size and frequency measure  $R_{sf}$  for reuse within single systems engineering from [5] leads to a value of  $R_{sf} = (63 - 27) / 63 = 0.57$ . This means that 57% of the needed artifacts are created by reuse. Thus, we have a

benefit of 57% for the considered part of the system compared to single system development.

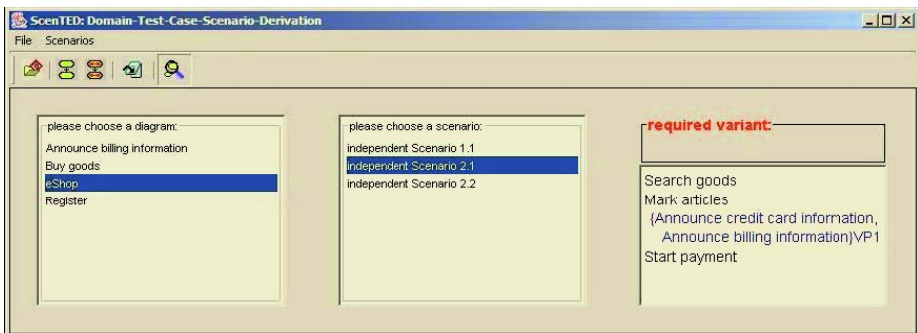
A questionnaire was used to get an estimation of the test engineers' opinion regarding the support of reuse offered by ScenTED. Statistical analysis has shown that ScenTED really supports the creation and reuse of test case scenarios. Details regarding the analysis of the questionnaire can be found in [18].

The case study stresses that the ScenTED technique reaches the goal of reduction in test creation effort. Due to the derivation and reuse of domain test case scenarios reduced test creation effort was achieved.

## 5 Tool Support for ScenTED

During the case study, the need for tool support was raised by the test engineers from Siemens. Especially the derivation of test case scenarios from activity diagrams was a time-consuming activity. The coverage had to be considered and navigation in the activity diagram hierarchy required concentration. Moreover, the retrieval of the application test case scenarios is cumbersome. Meanwhile, we have developed prototypical tool support for the ScenTED technique. We focused on the automated derivation of test case scenarios in domain engineering.

We have adapted an existing algorithm for branch coverage. This algorithm has been extended by the different cases specified in Section 3.1, i.e. the mandatory and optional as well as dependent and independent variation points. The formalization of the product family branch coverage criterion and its implementation led to the ScenTED-DTCD tool (Domain Test Case Scenario Derivation) as depicted in Fig. 9.



**Fig. 9.** Tool Support for Derivation of Domain Test Case Scenarios

The input for the tool is the mdl-file created by Rational Rose. The tool is able to deal with variation points and variants as far as they are specified as described in Section 3.1. The outputs of the tool are test case scenarios that describe the test engineer's activities and the system's response if modeled within the activity diagram. These scenarios are shown in the right section of this tool, but may also be saved

within a text file. An extension allowing the export of test case scenarios to a test case administration tool is currently under construction.

The screenshot in Fig. 9 shows the derivation test case scenarios for the eShop example. Three scenarios cover all branches of the example (see Fig. 5). The variability is preserved within the scenarios as shown in the right part of the image. The depicted scenario is one end-to-end scenario. All derived scenarios are independent scenarios due to the given example; therefore the field *required variant* is empty.

The implementation of the tool shows the formalization of the coverage criterion. We have explicitly defined variability in models and considered variability within test case scenario derivation. Model-based testing enables automation and the tool shows that this advantage is still available after the extension to software product family engineering.

## 6 Summary and Outlook

Testing is a serious problem in product family engineering. The productivity gains that have been realized for the constructive development phases could not be realized for testing until now. Moreover, the variability of the product family complicates testing in domain engineering.

In this paper we have presented the ScenTED technique for the system test of software product families. The contribution of this paper is the adaptation of modelbased testing to product family engineering: Test artifacts in domain engineering are extended to represent variability, in particular activity and sequence diagrams. The activities to derive test cases are enriched by an adapted branch coverage criterion for product family engineering and the binding of variants in test case scenarios.

ScenTED leads to reduction of test case development effort. In a case study within a recent project at Siemens Medical Solutions, the reuse benefit was up to 57% compared to application of single system testing techniques. Due to this success Siemens Medical Solutions HS IM adapted their testing process. ScenTED was put into daily practice and it is currently propagated to other departments at Siemens.

The advantages of model-based testing are realized by our technique, esp. early validation and automated test generation. During the application of ScenTED at Siemens, it was observed that requirements and in particular the variability within the requirements could be validated early due to the creation of the domain test case scenarios. Moreover, we demonstrated using a prototypical tool that the coverage criterion can be formalized and implemented.

Our future work aims at improving the technique and its tool support. We plan to improve ScenTED to save even more effort in product family testing. Effort can be saved in test case execution, if only those test cases are executed that test differences to a previously tested application. Even though there is a prototypical implementation, the technique lacks integrated tool support. The test models can be created in commercial tools like RationalRose or TestDirector, but creating the activity diagrams



as well as reusing test case scenarios for application testing has still to be performed manually. Our current work aims at providing full tool support for ScenTED.

## Acknowledgements

We would like to thank the staff at Siemens Medical Solutions HS IM that ontributed to the case study, especially Helmut Goetz, Frank Rometsch, Juergen Neumann, and Harald Lauritsch. Last, but not least we would also like to thank Thomas Rinke for his support in the creation of the prototype.

## References

1. Beizer, B.; "Black box testing", Van Nostrand Reinold, New York, 1990.
2. Bertolino, A.; Gnesi, S.; "PLUTO: A Test Methodology for Product Families", 5th Intl. Workshop on Product Family Engineering (PFE-5), Siena, Italy, November 2003.
3. Binder, R.; "Testing Object-Oriented Systems: Models, Patterns, and Tools", Addison-Wesley, Reading, 2000.
4. Clemens, P.; Northrop, L.; "Software Product Lines: Practices and Patterns", Addison-Wesley, Reading, 2002.
5. Devanbu, P.; Karstu, S.; Melo, W.; Thomas, W.; "Analytical and Empirical Evaluation of Software Reuse Metrics", 18th Intl. Conference on Software Engineering (ICSE), pp. 189-199, July 1995.
6. El-Far, I. K.; "Enjoying the Perks of Model-Based Testing", Software Testing, Analysis, and Review Conference (STARWEST 2001), 2001.
7. Geppert, B.; Li, J.; Roessler, F.; Weiss, D.; "Towards Generating Acceptance Tests for Product Lines", 8th Intl. Conference on Software Reuse 2004, Madrid, Spain, Springer, New York, pp. 35-48, 2004.
8. Halmans, G.; Pohl, K.; "Communicating the Variability of a Software Product Family to Customers", Software and Systems Modeling (SoSyM), Vol. 2, pp. 15-36, Springer, Hamburg, March 2003.
9. Hartmann, J.; Vieira, M.; Foster, H.; Ruder, A.; "TDE/UML: A UML-based Test Generator to Support System Testing"; 5th Annual International Software Testing Conference in India, 2005.
10. Hartmann, J.; Vieira, M.; Ruder, A.; "UML-based Approach for Validating Product Lines", Intl. Workshop on Software Product Line Testing (SPLiT), Avaya Labs Technical Report, pp. 58-64, Boston, USA, August 2004.
11. Hauber, R.; Ziegler, M.; Erskine, M.; Hilsenbeck, R.; "Modellbasiertes Testen", Objektspectrum, No 3, pp. 20 – 24, 2003 (in German).
12. Kamsties, E.; Pohl, K.; Reis, S.; Reuys, A.; "Testing Variabilities in Use Case Models", 5th Intl. Workshop on Product Family Engineering (PFE-5), Siena, Italy, November 2003.
13. McGregor, J.; "Testing a Software Product Line", Technical Report CMU/SEI-2001-TR-022, December 2001.
14. McGregor, J.; Northrop, L.; Jarrad, S.; Pohl, K.; "Initiating Software Product Lines", IEEE Software, Vol. 19, No. 4, pp. 24-27, July/August 2002.
15. Myers G.; "The Art of Software Testing", Wiley, New York, 1979.

16. Nebut, C.; Fleurey, F.; Le Traon, Y.; Jézéquel, J.-M.; "A Requirement-based Approach to Test Product Families", 5th Intl. Workshop on Product Family Engineering (PFE-5), Siena, Italy, November 2003.
17. Offutt, J.; Abdurazik, A.; "Generating Tests from UML Specifications", 2nd Intl. Conference on UML'99, 1999.
18. Reuys, A.; Goetz, H.; Neumann, J.; Weingaertner, J.; "Medizintechnik bei Siemens AG Medical Solutions HS IM", In: Boeckle, G.; Knauber, P.; Pohl, K.; Schmid, K. (eds); "Software-Produktlinien: Methoden, Einführung und Praxis", pp. 247-259, dpunkt, Heidelberg, 2004 (in German).
19. Reuys, A.; Reis, S.; Kamsties, E.; Pohl, K.; "Derivation of Domain Test Scenarios from Activity Diagrams"; Intl. Workshop on Product Line Engineering The Early Steps: Planning, Modeling, and Managing (PLEES'03), Erfurt, Germany, September 2003.
20. Riebisch, M.; Boellert, K.; Streidtferdt, D.; Franczyk, B.; "Extending the UML to Model System Families", World Conference on Integrated Design and Process Technology (IDPT 2000), Dallas, USA, June 2000.
21. van der Linden, F.; "Software Product Families in Europe: The Esaps & Café Projects", IEEE Software, Vol. 19, No. 4, pp. 41-49, July/August 2002.

# Quality-Based Software Reuse

Julio Cesar Sampaio do Prado Leite<sup>1</sup>, Yijun Yu<sup>2</sup>, Lin Liu<sup>3</sup>,  
Eric S.K. Yu<sup>2</sup>, and John Mylopoulos<sup>2</sup>

<sup>1</sup> Departamento de Informatica,  
Pontifícia Universidade Católica do Rio de Janeiro,  
RJ 22453-900, Brasil

<sup>2</sup> Department of Computer Science,  
University of Toronto,  
M5S 3E4 Canada

<sup>3</sup> School of Software, Tsinghua University,  
Beijing, 100084, China

**Abstract.** Work in software reuse focuses on reusing artifacts. In this context, finding a reusable artifact is driven by a desired functionality. This paper proposes a change to this common view. We argue that it is possible and necessary to also look at reuse from a non-functional (quality) perspective. Combining ideas from reuse, from goal-oriented requirements, from aspect-oriented programming and quality management, we obtain a goal-driven process to enable the quality-based reusability.

## 1 Introduction

Software reuse has been a lofty goal for Software Engineering (SE) research and practice, as a means to reduced development costs<sup>1</sup> and improved quality. The past decade has seen considerable progress in fulfilling this goal, both with respect to research ideas and industrial practices (e.g., [1, 2, 3]).

Current reuse techniques focus on the reuse of software artifacts on the basis of desired functionality. However, non-functional properties (qualities) of a software system are also crucial. Systems fail because of inadequate performance, security, reliability, usability, or precision, to name a few. Quality concerns, therefore, should also be front and centre in methods for software reuse. For example, in designing for the NASA Mars Spirit spacecraft, one would not adopt a “cosine” function from an arbitrary mathematical library. Instead, one must look for, and possibly adopt, a reusable component that meets stringent requirements in precision, performance and reliability.

Despite this practical need, few methods for reuse have focused on non-functional requirements (NFRs). The typical object of software reuse as surveyed in [1], is an artifact, initially executable code, and more recently large-scale components, software

---

<sup>1</sup> Improved software productivity and reduced development costs result from building *with* reuse; building *for* reuse actually has an overhead cost.

architectures, designs, frameworks, and software product lines. All of these are predominantly reused on the basis of functionality. One will not find precision, performance or reliability as components ready-made for reuse. Needless to say, it will be invaluable to reuse the knowledge about these critical requirements accumulated from the design of software for other spacecrafts, or from other domains.

Why is it hard to incorporate quality requirements into reuse methods? One important reason for this is that software artifacts include both functional and quality fragments. Some of the quality fragments are hard to recognize since they are mingled with the functional fragments in order to be executable.

Our goal is to focus on qualities as reusable assets. Would it be possible to separate knowledge about how to achieve a quality, such as “performance” from a specific function, say “cosine”? Would it be possible and reasonable to look for knowledge on performance, instead of looking for different implementations of “cosine”? Would it be possible to retrieve useful knowledge relative to a concern that is applicable in several domains? This is exactly the issue that we want to address in this paper.

Unfortunately, the cross-cutting nature of quality attributes in software makes them hard to classify. Cataloging them in terms of taxonomies [4, 5] is not sufficient support for proper software quality reuse. On the other hand, in traditional function-oriented classification, quality information is not discernible in the reusable artifact. This difficulty increases when there are multiple quality concerns being dealt within one artifact, which is often the case.

To overcome these difficulties, we combine insights and techniques from research in non-functional requirements [6, 7], goal-oriented requirements engineering [8], aspect-oriented programming [9], software reuse [1] and quality management. In particular we rely on the results of combining aspect-oriented programming with goal-oriented requirements engineering [10].

This combination proves to be effective because it unites a goal refinement and classification strategy with a packing strategy provided by aspect-oriented programming, making use of well-defined relations among functional and quality fragments, we provide mechanisms for weaving those fragments together. We define a coherent process that uses an asset library to find quality characteristics and apply those to a software functional description. We show that it is possible to store qualities, retrieve it for reuse, specialize it for different contexts and integrate it with functional descriptions. The process works both for graphs holding implementation information, as the detailed operationalization of goals (into tasks), or for graphs without such detail. The reuse process is therefore applicable at requirements as well as implementation level.

The paper is organized as follows. Section 2 provides some background on the concepts of goal-oriented requirements, on aspect-oriented programming, and on reuse. Section 3 reviews the obstacles to quality reusability, presents the concept of goal aspects, and proposes a language to support software quality reuse. Section 4 describes the overall process we foresee for automating part of the quality reuse process. Section 5 illustrates the process by demonstrating how the quality Usability can be reused. We conclude in Section 6 by noting the contributions and limitations of this work, positioning it with respect to the literature, and discuss future directions.

## 2 Goals, Aspects and Reuse

As stated in the introduction, we are using insights from different areas of research in software engineering as well as in management. Our aim is to provide means to make it possible to reuse software quality.

The starting point of our approach is the work on non-functional requirements [7] (NFRs) that treats them as softgoals in a goal dependency graph. This graph depicts interactions among goals where one goal can influence positively or negatively other goals. The similarity among NFRs and the concepts of aspect-oriented programming has been discussed in the literature and was exploited in [10]. Since both lines of work structure software with respect to qualities, they were central to our approach which uses goal graphs as a medium to organize software in relation to qualities, and in relation to functionality.

**Goals.** A goal represents a stakeholder intention. A goal can be either fulfilled or not [11], and may depend on sub-goals through AND or OR refinements. Goal-oriented requirements engineering [8, 11] focuses on goals which are “roughly speaking, precursors of requirements” [12]. Some goal-based modeling approaches, such as  $i^*$  [13, 14], also model the actors who hold these intentions.

Most variations of goal models in the literature use AND/OR trees to represent goal decomposition [6, 11] and define a space of alternative solutions to the problem of satisfying a root-level goal. There are several proposals for goal analysis techniques. For example, obstacle analysis [15] explores possible obstacles to the satisfaction of a goal. Along a different dimension, qualitative goal analysis [16] allows qualitative contributions from one goal to another, and shows how to formalize and reason with them. In whatever form, goal-oriented requirements engineering has been attracting considerable attention within the software engineering community [17, 18, 19, 20].

In [6], the concept of a softgoal has been proposed as a means for modeling and analyzing NFRs. Softgoals, unlike goals, can be partially satisfied or denied, and may depend on other goals and softgoals through Make(++), Help(+), Hurt(-), and Break(--) relations, also known as *contribution links* [6]. With goal models, software development proceeds by refining goals, identifying collections of leaf goals that together fulfill root-level goals, and assigning responsibilities for the fulfillment of leaf-level goals. Figure 1 provides an example of a goal model. In this paper, we use an ellipse, a rectangle and a cloud to represent a goal, a task, and a softgoal (quality) respectively. Each node has a type and a topic. A type describes a generic function or a generic NFR (a quality attribute). A topic, denoted in between “[” and “]”, describes contextual information. For instance, the goal “contact” refers to a “friend”, the softgoal “reliable” refers to “reply”.

**Aspects.** Factoring or factorization involves the decomposition of an object into a structure of smaller objects, or factors, which when combined together give the original. For example, the number 15 factors into primes as  $3 \times 5$ ; and the polynomial  $x^2 - 1$  factors as  $(x - 1)(x + 1)$ . The principle has been echoed in the software refactoring community [21] where refactoring is the process of rewriting material to improve its readability or structure, while preserving its meaning or behavior. For example, refactoring  $x^2 - 1$  as  $(x + 1)(x - 1)$ , reveals an internal structure that was previously not visible (such as the two roots of the polynomial at +1 and -1). Similarly, in software

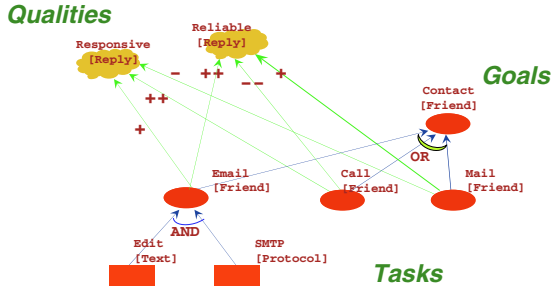


Fig. 1. A requirements goal model

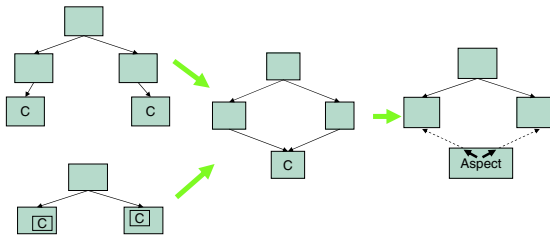


Fig. 2. Factoring principle in structured design and AOP

refactoring, the change in visible structure can often reveal the “hidden” internal structure of the original code. Extracting commonalities can also simplify the representation of potentially complex artifacts, e.g. re-expressing  $20 + 20 + 20$  as  $(1 + 1 + 1) \times 20$  or  $3 \times 20$ .

Aspect-Oriented Programming (AOP) [9, 22, 23] gives a different perspective to the factoring principle. Figure 2 illustrates how AOP is related to, and yet different from Structured Design [24]. The commonality may be modularized or refactored into a module *C* to avoid duplication. In Structured Design it is the responsibility of the user of a function to hold the address/name of the called module, whereas in AOP the responsibility of knowing where an aspect is needed relies on the aspect. One of the goals in structured design is to increase the fan-in of a module, a motivation shared with reuse.

An aspect [9] names the address of where it is needed as a pointcut. Pointcuts are not absolute addresses; they are virtual ones, making it possible that an aspect be applied in several places where the same conditions apply. An aspect keeps the information of what it does, as an advice. Through separation of crosscutting concerns, aspect-oriented languages offer simpler and more readable code structures. In order to execute the factored code, aspect-oriented environments use a reverse process known as weaving.

It is important to stress that the factoring principle as implemented in structured design and in aspect-oriented programming is to help reuse by consolidating similar information in just one place, thus making it easy to store and retrieve information.

An example aspect expressed in AspectJ syntax is as follows:

```
aspect DisplayUpdating {
    pointcut move(): call(void FigureElement.moveBy(int, int)) ||
        call(void Line.setP1(Point)) || call(void Line.setP2(Point)) ||
        call(void Point.setX(int)) || call(void Point.setY(int));
    after() returning: move() { Display.update(); }
```

The aspect `DisplayUpdating` includes the advice `Display.update()` that will be woven into the component code after the `move()` pointcut. A pointcut is a virtual address for the inclusion of the advice in a component. This virtual address is resolved through matching. For example, every time a `Line.setP1(Point)` appears in a component, the advice, `Display.update()` will be woven in that component.

**Reuse.** Aspects that crosscut different parts of the system arise likely to address global concerns of quality attributes represented by softgoals in the requirements goal model. The link among softgoals and aspects brings the possibility of using these concepts as basic entities to represent and organize qualities. On top of that we use a framework from quality management to better organize qualities.

We are anchoring our understanding of software reusability in Krueger's taxonomy for software reuse processes [1]. Krueger lists five key processes that should happen for a software artifact to be reused: classification, abstraction, selection, specialization and integration. Classification organizes the stored information to help future queries and updates, both by those who build for reuse, and by those who build with reuse. Abstraction helps understandability by hiding low-level details and implementation. Selection is the process where the actor building with reuse chooses what to reuse from the available reusable artifacts. Specialization is necessary in white box reuse, where an artifact needs to be changed to become reusable. To contrast, in black box reuse, an artifact is used as is. Finally, integration is necessary to make the artifact being reused fit into the context where it is going to operate.

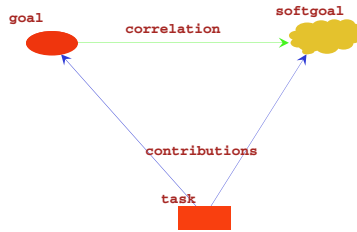
Although this process taxonomy is primarily concerned with functional reuse, we will use it to highlight the obstacles facing the reuse of qualities next.

### 3 A Goal- and Aspect- Driven Representation

This section outlines the key challenges when attempting to reuse qualities. We conclude that a better representation language is needed to achieve an effective reuse process. The primary insight is that a goal based representation allows qualities to be formally related to functional tasks through softgoal refinements and operationalizations.

#### 3.1 Obstacles to Reusing Software Qualities

As we have noted before, cataloging quality requirements as taxonomies are not yet sufficient to support proper quality reuse as it is not clear about which functionalities are bound to the quality concerns. On the other hand, software representation languages are known to lack non-functional concepts [12], which makes NFRs hard to be traced in the different representations used along the software construction process. As quality



**Fig. 3.** A V-shape goal model

concerns impact both high-level architectural changes and low-level code changes, they create difficulties in reuse when different levels of abstraction are related.

The selection of a particular incarnation of a given NFR is possible only if there is a way of linking the different incarnations with the required NFR. Since these incarnations are embedded in functional implementations, we also need to know how much these implementations satisfy<sup>2</sup> the given NFR. As such selection, from the non-functional perspective is a problem if the proper linkage and correlations among functions and qualities are not bound together.

We do not see an easy way for black-box reuse in the context of quality reusability. The key aspect in reusing qualities is how the selected instance of a given quality will be specialized into a new context. Specialization of a quality concern is hard, mainly due to its cross-cutting characteristic.

Last but not least, integrating a quality concern that was selected and specialized is another obstacle. The need for well-defined interfaces among the reusable and the new context is more complex than when dealing with functional concerns only.

### 3.2 Goal Aspects

Goal aspects were proposed in [10] to relate goal models representing functional requirements to softgoal models representing NFRs. Goals, softgoals and tasks are related by means of a V-graph, which is a graph with an overall shape of the letter V representing the three types of nodes (Figure 3).

The top two vertices of the V represent respectively functional and non-functional requirements in terms of goal models. Following [7] we represent NFRs in terms of softgoals, i.e., goals with no clear-cut satisfaction. Both models are AND/OR trees with lateral correlation links. The bottom vertex of the “V” represents a set of tasks that contribute to the satisfaction of both goals and softgoals.

A systematic requirements engineering process [10] uses the V-graph to elicit aspects, as in AOP terminology. We call these aspects, goal aspects, since they simplify the V-graph by removing the correlation links and putting functional and non-functional issues into separate AND/OR decomposition hierarchies. We have used it in the Media

<sup>2</sup> Herbert Simon [25] used the term *satisfice* to denote the idea of “good enough” solutions to an untractable problem. The NFR framework [7] is founded on the premise that NFRs (softgoals) are “satisficed” when they admit a partial, but good enough solution.



Shop case study [26]. The advantage of having a systematic process for discovering goal aspects is that finding them early on, makes it easier to trace quality concerns to aspect-oriented implementations. Although the V-graph representation helps the traceability of requirements and as such helps the processes of integration, specialization and selection, it does not fully support the classification and abstraction processes that are necessary for reusability. Next, we detail our proposal for a goal-oriented representation language to support quality reusability.

### 3.3 Q7: A Language for Organizing Qualities

As we have seen before, one of the key challenges in quality reusability is the multi-dimensional characteristic of quality issues. Classification of quality requirements and abstraction mechanisms to deal with them are obstacles to be overcome. These would require a language that could handle not only the characteristics of the quality knowledge, but that could relate those with functional descriptions as well. As such, we would need proper representation for the following concepts: functions, topics, quality types, pre-conditions, pointcuts (relations among functions, topics and quality types), contribution structures and quality operationalizations.

The source of inspiration for coming up with the abstract language was an analogy involving natural science and automobile design. In designing a sports car, a dominant quality to strive for is speed. If we think about speed in the context of marine life, we will observe that the fastest swimming animals have a common streamline shape. Further we will recognize that the streamline shape is manifested at different parts of the animal: the tail, the body and the front. If we based our automobile design on this concept we would need a car that would have special attention to the shapes of the rear part, the body and the front part. Although there is a huge gap in “reusing” this shape information, the analogy helped us in understanding that to locate a quality issue we would need to know why we need it, where it is applicable, and how to implement it. So, in the car as in the fish, when we need speed, the quality of speed needs to be applied to different parts of the fish or car, which when operationalized, are implemented as streamline shapes.

We could paraphrase the above as: having a reason (that is why), a place to apply the reason (where), and the details of the implementation to attain the reason (which is how). Once we made this connection, it came to us that the structure of the 5W2H used in quality management could be useful in classifying qualities.

Let’s see how the 5W2H fits into our context of quality reusability by examining each of the 7 questions (also known as Q7).

- *Why?* This question is central to a quality view; it addresses intentionality and focuses on the rationale of an intention. Non-functional requirements was initially proposed to describe quality attributes [6] to answer “Why an artifact needs a quality attribute?”. So the “why” question refers to the soft-goal or the quality information we want to reuse. In the NFR framework this is also known as “type”.
- *Who?* The “who” characterizes the main target of the quality attribute. In our analogy the fish and the car would be the target or the artifact to receive the speed attribute. So in our case, the “who” is representing the artifact associated with the soft-goal or the quality we would like to attain.

- *What?* The “what” characterizes contextual information of a given “who”, that will be the target of a quality attribute (“why”). It is a necessary triggering characteristic that the artifact must have to reuse the software quality. In the NFR framework this is also known as “topic”.
- *Where?* The “where” is the specific addresses of the quality concern in the artifact. In our V-graph goal model it is the pointcuts where the goal aspects will point to. This address is discovered by examining the correlations, in the NFR framework sense, found in a V-graph. “Where” is exactly the point in the V-graph that a goal aspect (“why”) will be woven. To be applied to this point the goal aspect has to comply with the “who” and “what” related to the reuse task at hand.
- *When?* The “when” is used to indicate a pre-condition that needs to hold before the operationalization (“how”) could be applied in a given pointcut (“where”). In the NFR framework is also known as a “claim”.
- *How?* This question addresses the refinement of the quality concern into a functional description. In the NFR framework this is known as the operationalization of a NFR [6]. It is how the NFRs will be implemented. In our model it will be the advices in our goal aspects.
- *How much?* The impacts and side-effects of applying the operationalizations (“how”) to the artifact. In the NFR framework it is the set of contributions links that relate the operationalizations with NFRs. Impacts can be implicit when they relate the operationalization and its parent softgoal.

Table 1 summarizes what is listed above and gives an example of the questions for the Media Shop case study.

Based on the above intuitions, we define the BNF grammar for the Q7 language, which organizes the knowledge for the purpose of software quality reuse.

```

START := Advice*
Advice := [When] [Who] Why [What] [Where] [How] [HowMuch]
When:= " (" Expr ")" "=>"
Who:= "<" id ">" " : ":"
Why:= id
What := "[" id { "," id }* "]"
Where:= "<=" Pointcut { "," Pointcut }*
How:= '{ ' BoolOp Advice* ' }'
HowMuch:= "=>" Effect { "," Effect }*
    
```

**Table 1.** Classifying the NFRs knowledge, such as the “Usability” aspect in Media Shop

artifacts	quality topics	quality types	claims	pointcuts	operationalizations	contributions
Who	What	Why	When	Where	How	How much
MediaShop	interface	usability	lang.	conventions	communicative	-productivity
MediaShop	interface	usability	operations	memorizability	operability	-productivity
MediaShop	interface	usability	usage	always	training	-productivity
MediaShop	lang.	communicative	words	natural lang.	lang. customization	-productivity
⋮	⋮	⋮	⋮	⋮	⋮	⋮

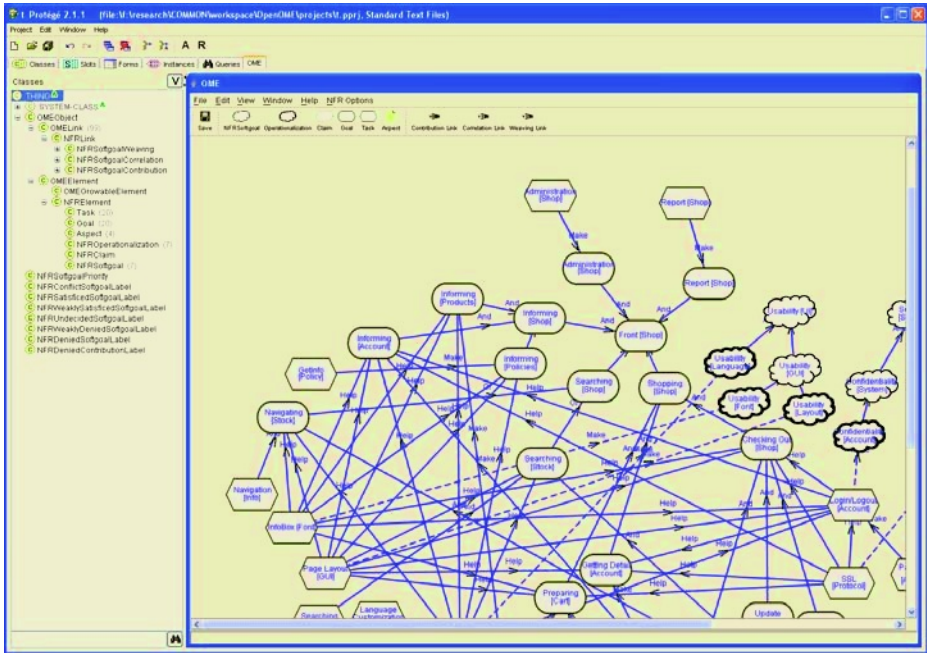
```
Expr:= "true" | "false" | id | Expr BoolOp Expr
Effect:= HowMuchOp [ Who "::" ] Why [ "[" What "]" ]
Pointcut:= HowMuchOp [ ("*"|Who) "::" ] ("*" | Why) [ ("[" "*" "]" | What ) ]
BoolOp := "&" | " | "
HowMuchOp:= "++" | "+" | "-" | "--"
```

We have designed a parser to convert a Q7 program into the Telos knowledge representation in our OpenOME tool.

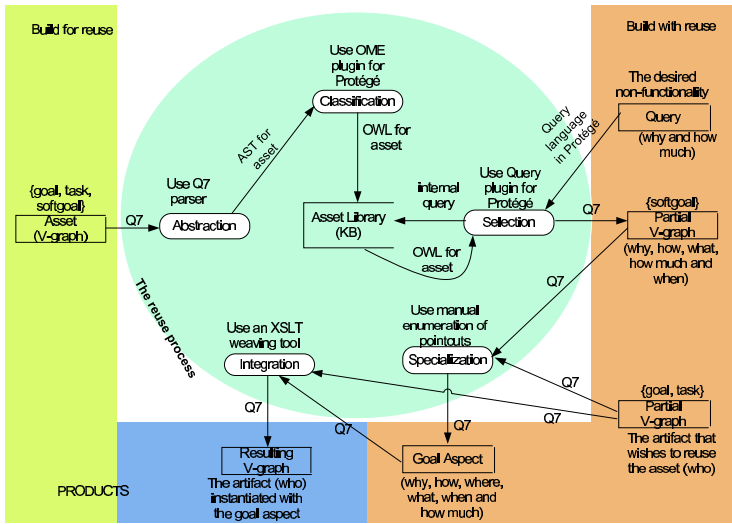
Next we describe how the Q7 language is central to our process for reusing qualities. If we look at Q7 just from the point of view of an organization scheme, it may look similar to a set of fixed facets. Faceted classification was proposed by Prieto-Diaz [27] to better organize a library of software components, where each component would have a description written as a set of facets. Although the facets may be defined at will, usually the examples shown in the literature did focus on the functional part of the components. Q7 goes beyond facets, by providing specific relationships among functional and quality concerns using an AND/OR graph as the basic representation scheme.

### 4 A Process for Quality-Based Reuse

Having framed the obstacles found in quality reuse in terms of the 5W2H framework, we now present a partially automated process to support quality reuse.



**Fig. 4.** The Media Shop build for reuse in OpenOME: the asset library is shown as the Protégé ontology to the left; the V-graph is viewed to the right



**Fig. 5.** A process for quality reusability

Figure 5 shows that our process is centered around the idea of an asset library. We have implemented this library by using a knowledge base approach with Protégé [28] on top of OWL [29, 30]. We have developed an OME/Telos (a tool for modeling NFRs) plugin for the Protégé 2.1.1 (Figure 4), which is capable of populating (TELL) and querying (ASK) the asset library. Figure 5 shows on the left hand side the input to the asset library – V-graphs for software systems (products) containing both functional and non-functional information, written in Q7. This description was produced from the point of view of building *for* reuse. The right part of the figure shows the products that are necessary in order to achieve quality reuse. We start with a general query on available assets for a given quality, retrieve a candidate for reuse, specialize the candidate by manual enumeration of pointcuts for a given functional description, and finally integrate the resulting goal aspects for the given functional description.

**Abstraction and Classification.** The processes of abstraction and classification are tasks for building for reuse, which we do not detail here. As said before we have developed basic infrastructure to support these tasks, by means of a parser for Q7 and the integration of Protégé with OME.

**Selection.** The selection process is performed by the software engineer using a Protégé plug-in to query the asset library. A query is performed to retrieve a needed quality that a developer is trying to reuse. This is done with the “why” operand, that is the query will return a partial V-graph with the softgoal sub-graph. This graph may be pruned by performing queries that narrow the search using the “when” and “how much” operands. It is possible to check for preconditions on the softgoal graph by the “when” operand, and to check for the satisficing levels (contribution links) of a given sub-graph by the “how” operands, to check the effects on other quality attributes by the “how much” operands.

**Specialization.** The process of specialization uses two V-graphs as inputs to produce the goal aspect graph for the chosen quality. The inputs are the V-graph retrieved from the selection process and the target V-graph, that is the V-graph representing the functional part where the quality has to be applied. As of now, we are using a manual inspection of both graphs to compose the resulting goal aspect graph. This task has to identify “where” in the functional representation the goal aspects must be woven. Doing this we are looking for the “what” or topic to which the advice of the goal aspect will be woven into. Note, that although Figure 5 does not explicitly have a feedback loop to the selection process, it occurs as we try to find a better candidate due to difficulties in the specialization process. We foresee several automation strategies to lessen the burden of a manual specialization: using matching patterns for topics in the inputs; using an automatic tool to locate bottlenecks of the artifact that needs operationalized quality improvements (e.g., using a profiling compiler that detects performance bottlenecks of the execution that needs the tuning advices); or using an external intelligent agent that performs the selection and specialization as a black-box reuse.

**Integration.** Once we have the V-graph for the selected and specialized goal aspect, we can use an automatic procedure, similar to an AOP weaver, to integrate the quality into the functional description. Unlike AOP, the weaving here not only insert the advices before/after/around the existing functionalities, but also allows a modification of the existing tasks by more advanced semantics, such as goal satisfactions, or function-preserving transformations.

In this paper, we studied a simple form of pointcuts specialization and weaving based on the goal satisfaction at the high-level requirements [10]. We believe it is extensible to low-level code weaving using existing tools such as AOP, compiler and transformation systems. The simple weaving is done as follows. The goal aspect V-graph produced by the specialization step and the functional V-graph provided by the software engineer are encoded in Q7, which are the inputs to the weaver, as described by the following procedure:

```

for each softgoal s
  P = WHERE.POINTCUTS(s)=test(WHEN.CONDITION(s), all functional goals)
  for each pointcut p of P
    for each functional goal g /* a candidate join point */
      if match(WHO(g), WHO(p)) and match(WHY(g), WHY(p))
        and match(WHAT(g), WHAT(p)) then
          weave(HOWMUCH.OP(p), HOW(g), HOW(s)) end if

```

The routine `test` uses the guard condition in the “when” clause to test whether any functional goal can apply the quality advice. These functional goals will be enumerated as the pointcuts in the quality aspect. To be more useful, these pointcut expressions can use wildcards to keep the virtual addresses.

The routines `match` checks whether a hard goal *g* matches the specifications of a pointcut softgoal *s*. They match if *g* has the same “who”, “why” and “what” as those of the pointcut of *s*. Wildcard “\*” in the pointcut specification can match any name.

The operation `weave` combines *g* and *s* using the pointcut operator (similar to “howmuch” operator), which is one of ++, +, - and --. First, a correlation link

$g \Rightarrow \text{op } s$  is created as an obligation on the joinpoint  $g$ . To fulfill this obligation, if the operator is  $++(--)$ , then all the subgoals of  $g$  (how) must add the operationalized tasks (anti-tasks) of  $s$  (how). If the operator is  $+( -)$ , then at least one of the subgoals of  $g$  must add the operationalized tasks (anti-tasks) of  $s$ . The semantic of the addition can be implemented using one of the “before”, “after”, “around” semantics in AOP.

In the next Section we show, with an example, how we have applied this process to retrieve usability from the asset library and reuse it in a different software system.

## 5 Reusing the Usability Asset

This section uses two software systems, Media Shop and Web Based Training (WBT), in order to illustrate the feasibility of our reuse process. The goal model describing the Media Shop case study was obtained using a goal aspect discovery process [10] and the goal model describing the WBT case study was obtained from an  $i^*$  model presented in [31]. Our aim is to reuse one of the qualities “usability” present in the Media Shop, and apply it to a different system – the WBT system.

For the Media Shop study we used both a requirements level description [26] and a real implementation, osCommerce [32], to trace the goals and softgoals to tasks and operationalized tasks. The goal aspect discovery process was applied on a V-graph merging the requirements level description with the recovered abstraction of the implementation. This V-graph is the asset we classified and stored in the asset library, which contains operationalizations for qualities such as security, usability, responsiveness and integrity [10]. An abstraction of the asset library is stored in the nested Q7 format. We only show the necessary parts for illustration purposes:

```
<MediaShop>::Front[Shop] { &
  ShoppingCart[Shop] { ShoppingCart[product, item] ... }
  Informing[Shop] { ... } Managing[Shop] { ... }
} => ++Security, ++Usability, ++Integrity, ++Responsiveness
Security[System] { ... }
Usability[UI] { & Usability[lang.] { & Communicative[Language] { &
  (NaturalLanguages) => LangCustomization[Words] <= ++<MediaShop>::ShoppingCart,
... } ... } ... } Integrity[Data] { ... } Responsiveness[Transaction] { ... }
```

We rewrote the functional part for the WBT system [31] using Q7. Below we list a partial description of the resulting goal model.

```
<WBT>::Build[System] { &
  CoursePattern[System] { |
    CoursePattern[InstructorLed] { &
      SchedulePresentation[Instructor] OptionalTopics[Learner]
    } CoursePattern[LearnerLed] { &
      ActAsLearningResource[Instructor] SetCoursePace[Learner]
    } } Collaboration[System] { |
    Collaboration[Email] Collaboration[NewsGroupForum] Collaboration
      [ChatRoom] Collaboration[SharingScreen] Collaboration[AVConf]
    } CommonLessonStructure[System] { |
```

```

Classic[Tutorial] ActivityCentered[Lessons] LearnerCustomized[Tutorial]
KnowledgePaced[Tutorial] Exploratory[Tutorial] Generated[Lessons]
}}

```

Given the functional description, our aim is to implement an interface for WBT that considers aspects of usability. Following the process in Figure 5, we reuse the usability asset as in our library. First we select from the asset library a softgoal hierarchy using a query (*why*="Usability"), resulting in an aspect without pointcuts in Q7:

```

Usability[UI] { &
  (Conventions) => Communicative[Language] { &
    (NaturalLanguage) => LangCustomization[Words]
  }
  (Conventions) => Communicative[Custom] { &
    (Classifications) => Customization[WordsOrder]
  }
  (Memorizability) => Operability[Operations] { &
    (MultipleWidgets) => Similar[LookAndFeel]
    (MultipleFonts) => Stylized[Font]
    (MultipleActions) => HierarchicalMenus[Navigation]
  }
  Training[Usage] { &
    ProvideUserManual[UseScenarios]
    ProvideContextSensitiveHelp[Actions]
    LearnByExamples[Tutorial]
  }
} } => -Productivity

```

For the root advice (*why*="Usability"), we have (*what*="UI", *when*="Conventions", *howmuch*="-Productivity"). The decomposition of the goal is nested inside the braces as detailed advices (*how*). We discard information stored in the asset library that is specific only to the asset. For example, (*where* = "ShoppingCart [product, item]") is a goal in the (*who*="MediaShop") domain that needs language customization for usability. To reuse the Usability in the "WBT" domain, however, "ShoppingCart" is irrelevant. Therefore we would only retrieve information that can be applied to any domains, such as (*when*="NaturalLanguage").

We perform the process of specializing the reusable asset as a goal aspect by updating the pointcuts. Currently, a human agent has to manually identify pointcut functional goals in the new domain according to the "when" condition in the queried aspect. For example, in WBT, any functional goal that involves "natural language" may consider the "language customization" advice. Therefore, one may enumerate the topics "Email", "ChatRoom", "NewsgroupForum", "Tutorial" and "Lessons" into the pointcut:

```

<WBT>::Usability[UI] { &
  (Conventions) => +Communicative[Language] { &
    (NaturalLanguage) => LangCustomization[Words] <= +++[Email],
    +++[ChatRoom], +++[NewsgroupForum], +++[Tutorial], +++[Lessons]
  }
  ... /* omitted */ } => -Productivity

```

The integration process is performed automatically and the resulting product is a Q7 description of WBT woven with the goal aspect of Usability. According to the Usability goal aspect, the operationalized task "LangCustomization[Words]" is only woven with the functional goals that match the pointcut.

```

<WBT>::Build[System] { &
  CoursePattern[System] { |
    CoursePattern[InstructorLed] { &
      SchedulePresentation[Instructor] OptionalTopics[Learner]
    } CoursePattern[LearnerLed] { &
      ActAsLearningResource[Instructor] SetCoursePace[Learner]
    } } Collaboration[System] { |
    Collaboration[Email] => ++LangCustomization[Words]
    Collaboration[NewsGroupForum] => ++LangCustomization[Words]
    Collaboration[ChatRoom] => ++LangCustomization[Words]
    Collaboration[SharingScreen] Collaboration[AVConf]
  } CommonLessonStructure[System] { |
    Classic[Tutorial] => ++LangCustomization[Words] }
    ActivityCentered[Lessons] => ++LangCustomization[Words]
    LearnerCustomized[Tutorial] => ++LangCustomization[Words]
    KnowledgePaced[Tutorial] => ++LangCustomization[Words]
    Exploratory[Tutorial] => ++LangCustomization[Words]
    Generated[Lessons] => ++LangCustomization[Words]
  } } => -Productivity

```

Since the V-graph of Media Shop also had the actual implemented goal aspects (given by osCommerce as explained in [32]) the final reuse will be the reuse of the code that implements the goal aspect, that is the desired quality. As such we rely on an operational semantics as to validate our final result. The fitness of the integration will depend on the quality of the specialization that was performed. Note that, by merging the two graphs, the semantics of the composed parts are preserved.

## 6 Conclusions

We have presented a method for making quality requirements a prominent dimension in software reuse. It is based on combining results from different research directions: requirements reuse and aspect-oriented programming

There are still several problems to be addressed, both from the point of view of supporting mechanisms as well as the feasibility of dealing with a large number of assets. The problem of scalability, dealing with huge graphs, is not itself the prime concern here, but how different strategies for partitioning the result of selection queries would be handled. It is also not clear where the strategy may break and how it will deal with very general quality concerns, for instance reusability. However this problem is general and also applies to the AOP view. Further research and experiments are needed.

As stated up front, reusing qualities is not an issue that received much attention in the literature. Two works, from different perspectives did approach the issue indirectly. One, [33], deals with the problem from the perspective of design patterns, while the other, [34], from the perspective of aspects. While Clarke and Walker [34] focus on parameterizing aspects to make them more flexible, Gross and Yu [33] propose to explicitly deal with quality concerns in design patterns and, as such, propose an explicit encoding of the intentionality for each pattern. In our proposal we provide a broader view of the problem and address all the five software reuse key processes.



## References

1. Krueger, C.: Software reuse. *ACM Computer Survey* **24** (1992) 131–183
2. Prieto-Diaz, R.: Status report: Software reusability. *IEEE Software* **10** (1993) 61–66
3. van Vliet, H.: *Software Engineering: principles and practice*, 2nd Ed. John Wiley (2000)
4. Sommerville, I.: *Software Engineering*, 4th Ed. Addison-Wesley (1992)
5. Boehm, B.W., Brown, J.R., Lipow, M.: Quantitative evaluation of software quality. In: ICSE, International Conference on Software Engineering, IEEE Computer Society Press (1976) 592–605
6. Mylopoulos, J., Chung, L., Nixon, B.: Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng.* **18** (1992) 483–497
7. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishing (2000)
8. Mylopoulos, J., Chung, L., Yu, E.: From object-oriented to goal-oriented requirements analysis. *CACM* **42** (1999) 31–37
9. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect oriented programming. *LNCS* **1241** (1997) 220–242
10. Yu, Y., do Prado Leite, J.C.S., Mylopoulos, J.: From goals to aspects: Discovering aspects from goal models. In: RE 2004 International Conference on Requirements Engineering, IEEE Computer Society Press (2004) 38–47
11. van Lamsweerde, A.: Goal-oriented requirements engineering: From system objectives to UML models to precise software specifications. In: ICSE 2003. International Conference on Software Engineering, IEEE Computer Society Press (2003) 744–745
12. Feather, M.S., Menzies, T., Connelly, J.R.: Relating practitioner needs to research activities. In: RE 2003. International Conference on Requirements Engineering, IEEE Computer Society Press (2003) 352–361
13. Yu, E.S.K., Mylopoulos, J.: From E-R to A-R – modelling strategic actor relationships for business process reengineering. *Int. Journal of Intelligent and Cooperative Information Systems* **4** (1995) 125–144
14. Liu, L., Yu, E., Mylopoulos, J.: Security and privacy requirements analysis within a social setting. In: RE 2003. International Conference on Requirements Engineering, IEEE Computer Society Press (2003) 151–161
15. van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Softw. Eng.* **26** (2000) 978–1005
16. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with goal models. *LNCS* **2503** (2002) 167–181
17. Anton, A.I., Carter, R.A., Dagnino, A., Dempster, J.H., Siegel, D.F.: Deriving goals from a use-case based requirements specification. *Requirement Engineering* **6** (2001) 63–73
18. Rolland, C., Prakash, N.: From conceptual modelling to requirements engineering. *Annals of Software Engineering* **10** (2000) 151–176
19. Kaiya, H., Horai, H., Saeki, M.: Agora: Attributed goal-oriented requirements analysis method. In: RE 2002. International Conference on Requirements Engineering, IEEE Computer Society Press (2002) 13–22
20. Bolchini, D., Paolini, P., Randazzo, G.: Adding hypermedia requirements to goal-driven analysis. In: RE 2003. International Conference on Requirements Engineering, IEEE Computer Society Press (2003) 127–137
21. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: *Refactoring: Improving the Design of Existing Code*. Addison-Wesley (1999)
22. Murphy, G.C., Walker, R.J., Baniassad, E.L.A., Robillard, M.P., Lai, A., Kersten, M.A.K.: Does aspect-oriented programming work? *CACM* **44** (2001) 75–77

23. Robillard, M.P., Murphy, G.C.: Concern graphs: finding and describing concerns using structural program dependencies. In: Proceedings of the 24th International Conference on Software Engineering (ICSE-02), New York, ACM Press (2002) 406–416
24. Yourdon, E., Constantine, L.L.: Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, 1st ed. Prentice-Hall (1979)
25. Simon, H.A.: The Science of the Artificial, 3rd Edition. MIT Press (1996)
26. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the tropos project. *Information Systems* **27** (2002) 365–389
27. Diaz, R.P.: Implementing faceted classification for software reuse. *Commun. ACM* **34** (1991) 88–97
28. Noy, N.F., Sintek, M., Decker, S., Crubezy, M., Ferguson, R.W., Musen, M.A.: Creating semantic web contents with Protege-2000. *IEEE Intelligent Systems* **16** (2001) 60–71
29. W3C: Web ontology language, <http://www.w3.org/2004/owl> (2004)
30. Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F.: OIL: an ontology infrastructure for the semantic web. *IEEE Intelligent Systems* **16** (2001) 38–45
31. Liu, L., Yu, E.: Design web-based systems in social context: A goal and scenario based approach. In: CAiSE 2002. Volume 2348., Springer-Verlag (2002) 37–51
32. : (Open Source E-Commerce Solutions, <http://www.oscommerce.com>)
33. Gross, D., Yu, E.S.K.: From Non-Functional Requirements to Design through Patterns. (Requirements Engineering)
34. Clarke, S., Walker, R.J.: Composition patterns: An approach to designing reusable aspects. In: ICSE 2001. International Conference on Software Engineering, IEEE Computer Society Press (2001) 5–14

# On the Lightweight Use of Goal-Oriented Models for Software Package Selection

Xavier Franch

Universitat Politècnica de Catalunya (UPC),  
UPC – Campus Nord, Omega-122, 08034 Barcelona (Spain)  
franch@lsi.upc.edu  
<http://www.lsi.upc.edu/~gessi>

**Abstract.** Software package selection can be seen as a process of matching the products available in the marketplace with the requirements stated by an organization. This process may involve hundreds of requirements and products and therefore we need a framework abstract enough to focus on the most important factors that influence the selection. Due to their strategic nature, goal-oriented models are good candidates to be used as a basis of such a framework. They have demonstrated their usefulness in contexts like early requirements engineering, organizational analysis and business process reengineering. In this paper, we identify three different types of goal-oriented models useful in the context of package selection when some assumptions hold. *Market segment models* provide a shared view to all the packages of the segment; *software package models* are derived from them. The selection can be seen as a process of matching among the *organizational model* and the other models; in our proposal this matching is lightweight, since no model checking is performed. We define our approach rigorously by means of a UML conceptual data model.

## 1 Introduction

In the last years, software-package (SP)-based information systems have become not the exception but the rule in the software-based solutions for managing the informational resources of organizations worldwide. This is true for both SP that have a visible impact in the services offered by the organization, such as ERP and CRM systems, and SP that take care of the daily functioning of the organization, such as mail or meeting scheduler systems, and security-related tools.

Successful SP-based system development requires a unique set of activities to be performed, among which we find the selection of the SP themselves. This activity is becoming increasingly more and more critical, due to the ever-growing nature of the SP market, both in the variety of market segments (MS) available and the SP offered therein. As a consequence, several SP selection methodologies, processes and techniques have been formulated [1, 2, 3, 4], which propose different ways of eliciting requirements and evaluating SP in the context of package selection. In spite of this variety, we think that these proposals do not address in an effective way 2 fundamental questions:

- *MS variety*: How can we describe the different existing MS in a way such that an organization becomes aware of their applicability in a particular selection problem?
- *SP proliferation*: How can we describe the great deal of SP belonging to a MS in a way such that their comparison can be made uniformly?

The answer to these questions should take into account the following facts:

- *Complexity of the problem*: Not only the SP market is large, also the number of requirements can be very high in typical selection processes for information systems, as well as the number of quality factors that characterise a particular kind of SP. As a consequence, we cannot aim at considering the detailed system requirements nor all the SP quality factors from the beginning.
- *Intertwining of requirements elicitation and SP evaluation*: We need ways to facilitate the identification and reformulation of requirements from the observation of the SP market.
- *Evolution of the SP market*: New SP, and versions of existing SP, appear continuously, preventing therefore the use of exhaustive descriptions of these SP.

A natural way for answering the raised questions considering the enumerated facts is to rely on goals in the early stages of SP selection, rather than on detailed requirements, specifications or quality models [5]. In fact, this idea aligns with the observation that current goal-oriented analysis methods and languages such as KAOS, *i\**, GRL or TROPOS [6, 7, 8, 9] are widespread in the requirements engineering community for the refinement and decomposition of the customer needs into concrete goals in the early phase of the requirements specification [10]. In our context, goals and goal-based analysis can be used to model organizational needs, to identify which MS are interesting for the problem at hand and to make a screening of the candidate SP belonging to these identified MS. Of course, detailed requirements and evaluation of interesting quality factors should be made, but at a later stage when the whole scenario has been clarified and the solution space has been pruned down to a small set of candidate SP.

Once the applicability of goals to SP selection has been stated, we face a problem. Existing goal-based methodologies for software systems construction (e.g., TROPOS, KAOS methodology) do not handle explicitly the particularities of SP selection. In other words, they seem to be addressed more to support the development of bespoke software rather than the construction of systems based on the composition of SP.

In this paper, we use of goal-oriented models for supporting SP selection. More precisely, we propose dependency-based goal-oriented models, i.e. models that state which are the actors involved in the information system and which actor goals depend on which other actors. We use *i\** Strategic Dependency models [7] with this purpose. We distinguish 3 types of models: 1) organizational models for modelling the needs of the organization; 2) MS models for modelling the services that a particular MS addresses; and 3) SP models for modelling the services that a SP belonging to a particular MS offers. We show how the relationships among these 3 models can be defined and how do they help to answer the 2 fundamental questions above taking the 3 identified facts into account (see fig. 1). We provide a rigorous description of all the models and their relationships through a UML conceptual model [11], including a complete set of constraints that are presented in textual form (see [12] for their OCL form).

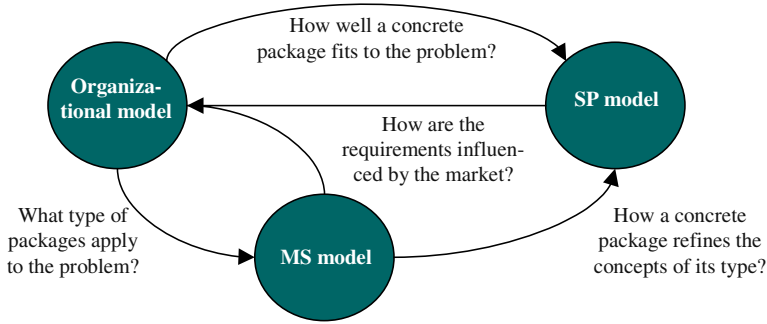


Fig. 1. Using goal-oriented models in the selection process

## 2 Scope of the Proposal

This research has been motivated by a collection of selection processes in which we have been recently involved, some of them reported elsewhere [13, 14, 15]. These experiences have resulted in the identification of the questions and facts presented in the introduction and they have yielded to the adoption of goal-oriented models as explained in this paper. Therefore, we may say that our proposal has been validated through experiments with the following assumptions:

- **Usefulness:** The MS addressed is a segment of general interest. This means that a great deal of organizations need to select SP from this MS. Some examples are: communication infrastructure (mail servers, videoconference, etc.), ERP systems, security-related systems, etc.
  - *Consequence:* The number of selection processes that take place in this MS will be high and then reusability of the models likely to occur.
- **Variety:** There are a lot of SP offered in this MS that are competitive enough. The first part of the statement is a direct consequence of the previous point: if there is a universal need, of course lots of products will be offered. The second point excludes some particular MS like operating systems in which options are few and the analysis we propose does not pay.
  - *Consequence:* The number of SP to be analysed in selection processes in this MS will be high and then we need a common framework as a basis for analysing and comparing them.
- **Size:** The type of SP offer a great deal of features. This makes SP understanding more difficult, time-consuming and cumbersome. SP such as ERP systems are typical examples, whilst time or currency converters are not.
  - *Consequence:* The concepts embraced by the type of SP are much and then light descriptions focusing in the most fundamental notions are needed for pruning the set of candidates in a cost-effective manner before detailed evaluations occur
- **Continuity:** The selection activity is monitored by an organization that accumulates experience from past selection processes. This organization will find

valuable to have means to transfer knowledge from one experience to another and to assist their clients in the maintenance of their SP-based information system.

- *Consequence*: The organization will be involved in an increasing number of selection processes being able to transfer knowledge while improving its skills.
- **Uncertainty**: The starting requirements stated by the organization are vague, incomplete and often ill-justified. Sometimes the organization even does not know exactly which MS is addressing to.
  - *Consequence*: The statement of the organizational departing needs must cope this incompleteness focusing in the strategic underlying needs instead of the concrete requirements. Furthermore, it should be possible to add new needs from the SP analysis, which means that both organizational and software models should be described similarly.

The use of our approach in SP selection processes with different assumptions would require further experimentation.

### 3 Background: Strategic Dependency Models in the $i^*$ Notation

An  $i^*$  *Strategic Dependency (SD) model* comprises two types of elements, *actors* and *dependencies* among them. Actors are intentional entities, that is, there is a rationale behind the activities that they carry out. Dependencies connect source and target actors, called *depender* and *dependee*. Altogether they form a network of knowledge that allows understanding “why” the system behaves in a particular way [16].

For our purposes, actors play roles that are abstract characterizations of a behaviour within some specialized context or domain of endeavour [7]. We distinguish four types of actors: human, organizational, software and hardware. We find convenient to allow the definition of hierarchies using the typical is-a construct.

SD models express dependencies using four main types of dependency link (see fig. 2). For *goal dependencies* the *depender* depends upon the *dependee* to bring about a certain state in the world. A *goal* represents a condition or state of the world that can be achieved or not. For *task dependencies*, the *depender* depends upon the *dependee* to carry out an task. A *task* is a detailed description of how to accomplish a goal. For *resource dependencies*, the *depender* depends upon a *dependee* for the availability of an entity. *Resources* can be physical or informational, and considered as the availability of some entity or even the finished product of some process. For *soft goal dependencies*, the *depender* depends upon the *dependee* to perform some task that meets a non-functional requirement, or to perform the task in a particular way.

Fig. 3 presents a the class diagram from a UML conceptual model that formalizes the concept of SD model; the integrity constraints required here are: (IC1) an actor shall not be a specialization of itself; (IC2) specialization shall preserve the type and model of an intentional element; (IC3) the depender and dependee of a dependency shall belong to the same model and shall not be the same neither one a specialization of the other. We include in the diagram the 3 specializations of SD models enumerated in the introduction that will be presented next.

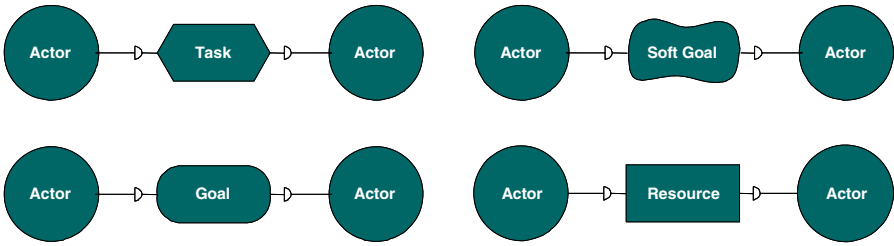


Fig. 2. Graphical representation of *i\** Strategic Dependency actors and dependencies

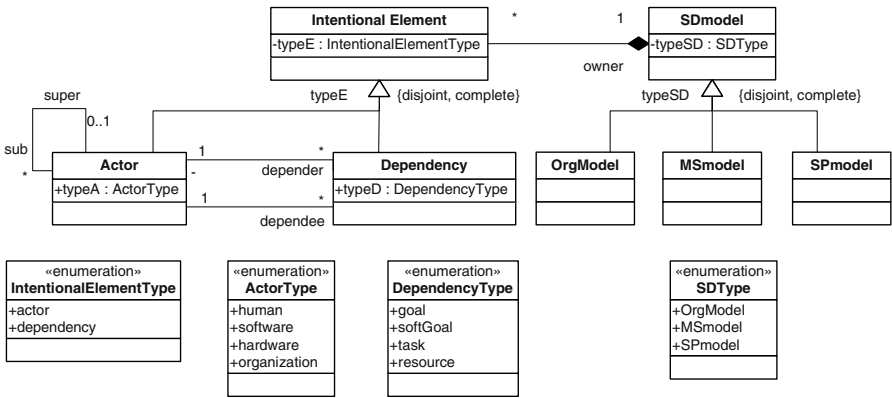


Fig. 3. A UML conceptual model representing SD models and their specializations

### 4 A Goal-Oriented Model for Stating Organizational Needs

As mentioned in the introduction, the classical use of goal-oriented models in the context of information systems is to provide an early specification of the system-to-be focusing on strategic aspects [5, 10]. We consider that the first set of tentative, incomplete and high-level requirements in a selection process is the formulation of such a model, which is obtained using some methods widespread in the requirements engineering community such as GBRAM [17]. Since this is not a contribution of our work, in this section we just introduce an example that we are going to use in the rest of the paper.

Let’s consider an organization concerned with ensuring data integrity when data interchange take place with human users. Fig. 4 presents its intentional elements. Users who interchange data require the organization to keep their information preserved (D1) and not to send them undesired information of any kind (D2). Since users are aware that they may submit incorrect information, they also require the organization to warn them in this case (D3). Information checking shall be transparent to users (D5). At its turn, the organization just requires users not to submit hazardous information of any kind (D4). On the other hand, the organization needs support to discern whether its managed information contains unwanted data or not (D6). This gives light to a third actor –a data integrity expert, capable of informing whether the information suffers from some hazards or not.

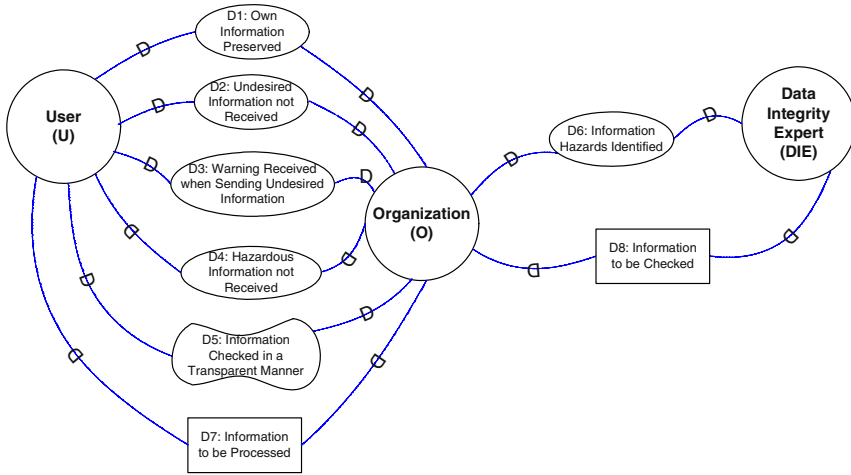


Fig. 4. SD model for an organization with data integrity needs

### 5 *i*\* SD Models for the Strategic Description of Market Segments

*i*\* SD models are appropriate artefacts to describe the services that SP belonging to a MS offer to organizations. When used with this purpose, we call them *market segment SD models*, *MS models* to abbreviate. MS models are characterised by the following properties (see fig. 5):

- There is a designated actor that represents the type of SP characterising a MS.
- There is at least another actor representing the main SP beneficiary, that most of the cases is an organization. Sometimes, this actor is specialized into different subactors using an “is-a” construct.
- Often there are some other actors bound to the type of SP of interest, for instance:
  - Abstract software actors, standing for software that may require connection to the type of SP of interest but that is unknown in advance.
  - Human users representing stakeholders have particular interests that are distinguishable from those organizational ones (e.g., end users).
  - SP administrator, in charge of administration duties when packages are large.
- The dependencies among the actors are kept to the minimum extent. There are two main reasons for this decision:
  - The model shall be general enough to be inclusive, i.e., every single SP of the modelled type must be compliant with that model.
  - SP selection will include a matching process involving this model, and in this context it is convenient to handle models of a reasonable size.
- Most dependencies are goal dependencies. Soft goal dependencies are restricted to crucial non-functional requirements on the type of SP. Resource depen-



dependencies are restricted to fundamental concepts of the MS that stand for some kind of data. Task dependencies are restricted to activities that are out of the control of the depender.

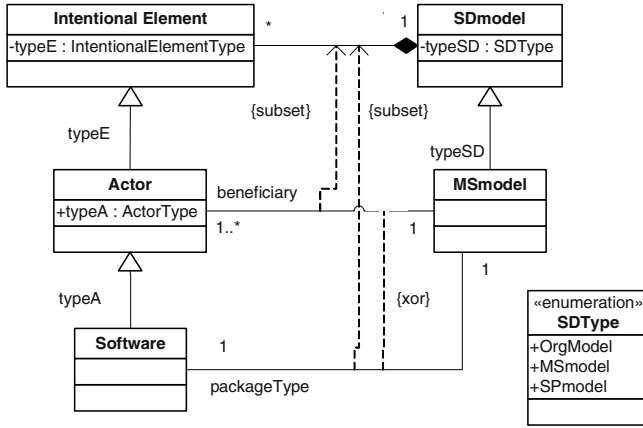


Fig. 5. A UML conceptual model representing MS models

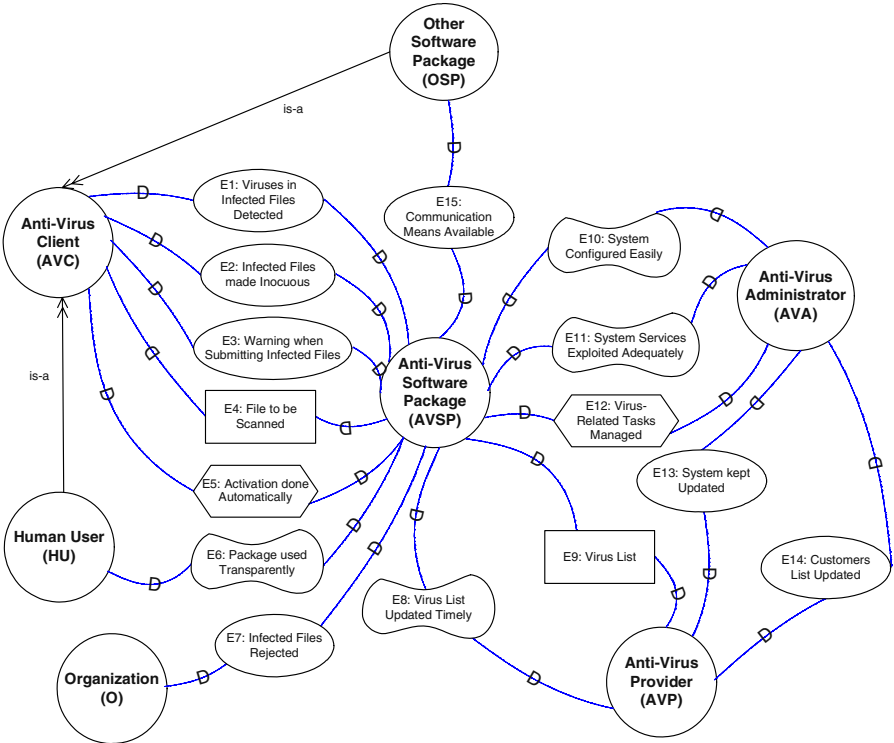


Fig. 6. An SD model for the market segment of anti-virus software packages

A MS that can be used for satisfying the organizational model presented in section 4 is the one of anti-virus software packages (AVSP) (see fig. 6). This segment fulfils the requirements outlined in section 2 in order to be a valid target of our proposal.

The guidelines concerning dependencies can be checked in the figure. For instance, some non-functional properties such as efficiency are not required at any point –of course we would like the solution to be efficient, but we consider that for AVSPs, efficiency is not as critical as for other types of packages and therefore it does not appear in this highly strategic model. The two resource dependencies show the two most significant data concepts: the target of the SP (the file to be scanned, E4) and the object that threatens the file (the viruses, E9). Task dependencies reveal that AVSP activation (E5) and task management (E12) occur in a particular manner.

## 6 Software Package Description Using $i^*$ SD Models

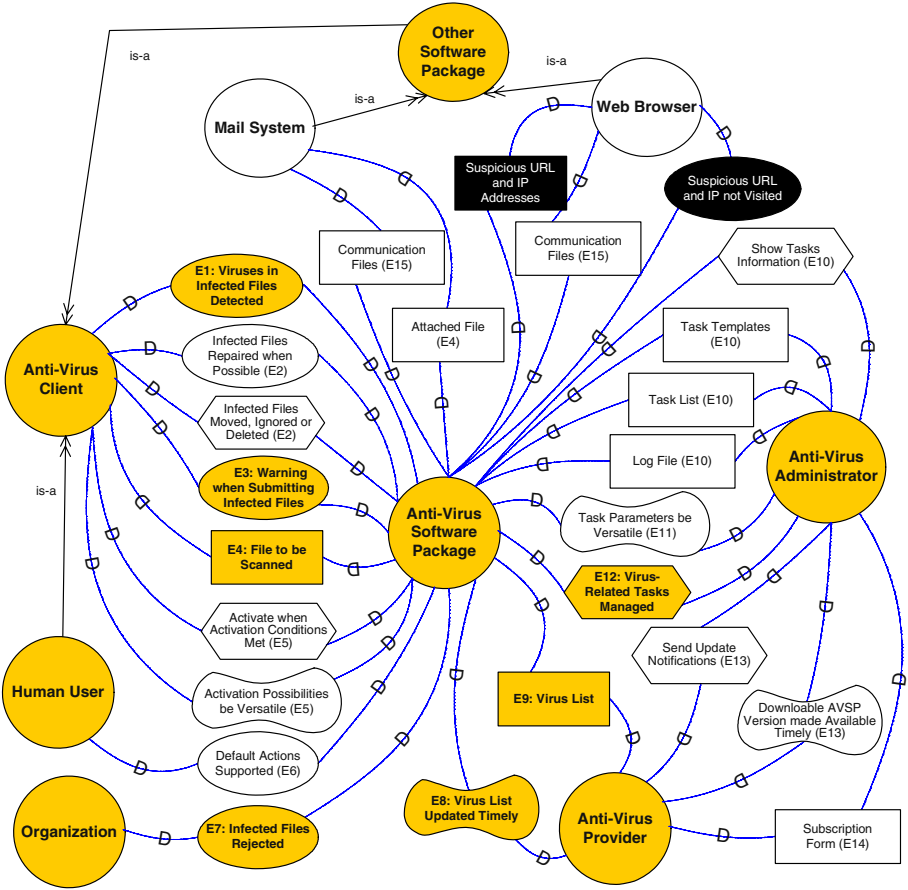
As mentioned, a MS model exhibits a fundamental property: every single SP of this segment shall be compliant with this model. This helps to tackle one of the fundamental questions identified in the introduction, namely SP proliferation. However, each package may have its own peculiarities with respect to the services offered to and requested from its environment. We propose to use again  $i^*$  SD models to represent them and, in particular, we claim that the description of a particular SP shall be build starting from the MS model. We call this model *software package SD model*, or *SP model*. The process of obtaining a SP model from a MS one is called *derivation*.

In SP models, intentional elements can be of 3 types depending on their relationship with MS models:

- *Kept elements* are those already appearing in the MS model. They stand for intentional elements that are not further refined for any of the following reasons:
  - They are resources or tasks detailed enough in the departing MS model.
  - Their refinement would not add strategic value to the selection process.
- *Refined elements* are those not present in the MS model but refining one or more of this model, which are called *refinable elements*. Refined elements express some MS concept in a more concrete way. Thus, some situations are not allowed, more precisely a task or resource dependency cannot be refined into a goal dependency. Non-kept SP actors that are defined as specializations also fall into this category, since in our context specialization may be conceptually considered as refinement.
- *New elements* are those not present in the MS model and not refining any of its elements. Usually they express advanced capabilities of a particular SP that are not standard in the corresponding MS.

As an example, we derive a SP model for a particular kind of AVSP, the McAfee VirusScan v. 4.5.1 [18], see fig. 7. Kept and refined dependencies identify which elements of the corresponding MS model for AVSP (cf. fig. 6) they correspond to. We introduce two new software actors inheriting from the “Other Software Package” actor, because the VirusScan supports connection with mail systems and web browsers. In both cases, communication is implemented via some files placed at some designated directories; this yields two refined dependencies. On the contrary, the

AVSP actor: in the case of the mail system, refines the concept of “File to be Scanned” into “Attached File”; in the case of web browser, requires a specification of the addresses that must be avoided in visits, which are new dependencies generated by the new actor. The other actors are kept from the MS model.



**Fig. 7.** An SD model for the VirusScan anti-virus SP (grey: kept elements; white: refined elements; black: new elements; enclosed in parenthesis, references to the MS model).

Fig. 8 models the concept of derivation as an association among MS and SP models. At its turn, the derivation concept can be defined as a set of links among elements from the MS and SP models<sup>1</sup>. To do so, elements of the involved models are bound to the association class and then put together with the *DerivationLink*

<sup>1</sup> We could have used a ternary association, but splitting into two binaries makes the integrity constraints easier to write, specially in their OCL form.

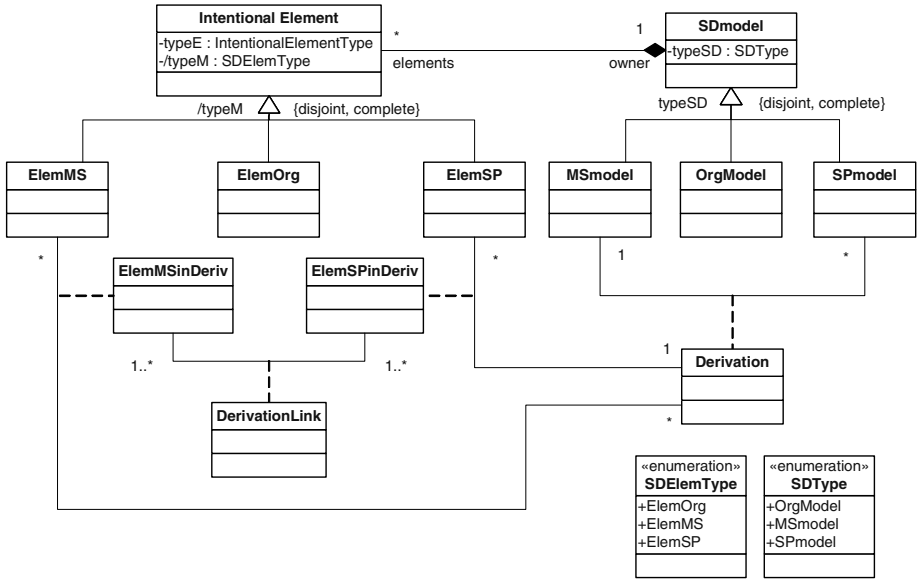


Fig. 8. Derivation from MS to SP models

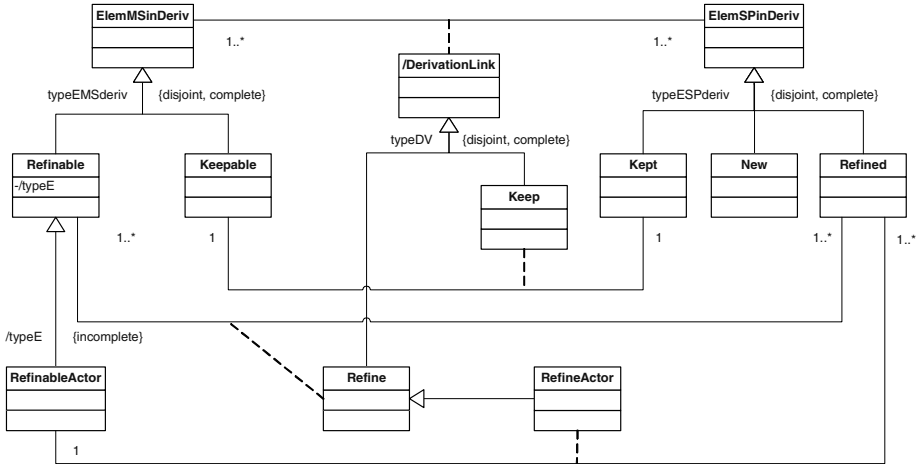


Fig. 9. Derivation from MS to SP models (detailed view).

association class. Five integrity constraints are required to state that the derivations shall preserve the type of their intentional element (IC4), the type of their actors (IC5), their specialization hierarchies (IC6) and also the actors involved in their dependencies (taking actor specialization into account; IC7), and shall belong to the same model (IC8). We need an additional constraint to state that derivations shall be complete (i.e., each element from the MS model shall be derived into the SP model);



Unlike this example, model matching will usually be incomplete and overloaded. In the case of MS, if the degree of incompleteness is too high, the MS will not apply to the problem. Otherwise, one or more of the following statements will hold: 1) the information-system-to-be will combine SP coming from different MS; 2) some glue code is necessary to achieve the expected functionalities; 3) uncovered organization requirements must be prioritised to decide if they can be relaxed; 4) the MS offer some functionalities or behavioural characteristics that were not foreseen by the initial requirements of the system and that can be incorporated. Even a complete matching does not exclude that other MS are also selected as applicable. In our case, in addition to AVSP, we could thought of data encryption and spyware tools MS as applicable.

The UML class diagram for specifying the matching concept is presented in fig. 10. There are some similarities with the specification of derivation presented in the previous section, both for the class diagram itself and the integrity constraints<sup>2</sup>: we remark that IC4 hold and also IC6, IC7 and IC8 but just partially: dependencies in the organizational model may be matched with dependencies in the MS or SP models among the same two actors up to specialization (and then IC6 to IC8 apply) or among one of the organizational actors and one the new actors (the system, the administrator, ...). There are not further restrictions about the type of actors or dependencies. On the other hand, both types of matching are closely related: we need an integrity constraint (IC11) stating that if an intentional element *A* belonging to an organizational model matches with an intentional element *B* belonging to a MS model and with an intentional element *C* belonging to a SP model, then *C* shall be derived from *B* (either kept or refined). We define as derived attributes the coverage of matching.

As already mentioned, the matching can be used to point out new organizational needs for the organization. In particular we may say the following:

- When matching with a MS model, it holds that: each dependency *D* in the MS model that links a new actor (one that does not match with a organizational actor) and a matched actor (one that matches with a organizational actor) is indicating some dependency that is not identified in the departing organizational model.
- When matching with a SP model, it holds that: for each dependency *D* in the SP model that links a new actor and a matched actor:
  - If *D* is derived or kept from a dependency *E* that appears in the MS model from which the SP model derives, then analyse the matching among the MS model and organizational model and behave as in the case before. We act this way because the MS model is more abstract and then closer to the organizational point of view.
  - Otherwise, the dependency *D* itself is considered.

On the other hand, actors in the MS and SP models that inherit from other actors that are matched to organizational one, are also worth to be considered for inclusion in the organizational model. This is the case of the mail system and web browser actors from the McAfee VirusScan model.

---

<sup>2</sup> In [12] we have defined some metaclasses that generalise the derivation and matching concepts and we define some of the associations and integrity constraints in the superclasses.

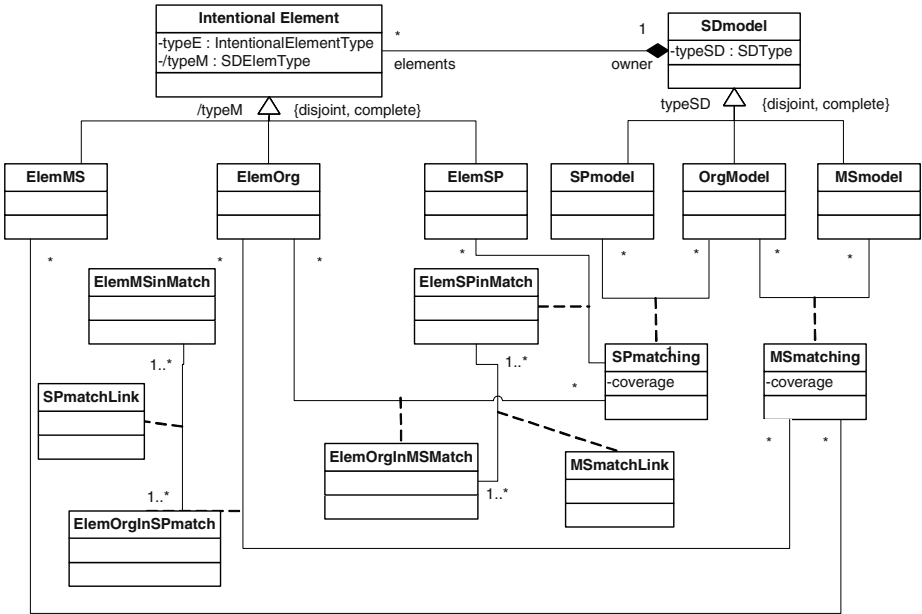


Fig. 10. Matching among organizational, MS and SP models.

Please note that we are not proposing automatic updating of the organization model, since this requires a careful strategic analysis. The contribution here is that we provide a systematic way to identify candidate modifications to be performed.

## 8 Conclusions

We have presented an approach for using a widespread specification technique such as goal-oriented modelling in the increasingly important context of software package selection. Our proposal is based on the notions of matching and derivation which bind intentional elements belonging to different goal-oriented models. We classify our approach as lightweight because these 2 kinds of correspondences have been defined without any type of model-checking techniques to validate that the elements involved satisfy some formula. We have provided a rigorous meaning to the proposal by means of a UML conceptual model. We believe that our proposal has a positive impact to both the context of SP selection and goal-oriented modelling. For SP selection:

- It provides a starting point for this activity, focusing on goals before more detailed issues such as measurable requirements.
- It acts as a high-level documentation for recording the most strategic decisions and services arisen in the selection process.
- It supports the classical SP-based software systems life cycle in which SP selection and requirements elicitation are two complementary activities.

- It supports transfer of knowledge (and therefore return on investment) from one selection process to other of the same MS.
- It fits well with the extreme volatility of the market, because of the traceability implied by the derivation and matching notions.

Concerning goal-oriented modelling, existing approaches such as TROPOS [9] are primary concerned with the use of goals for guiding software development. We think that our approach can be linked to TROPOS to obtain a slightly different methodology aimed at SP-based software development, in which the requirements elicitation phase may be defined using the concepts presented in this paper and system implementation may be seen mainly as a SP integration activity. Addressing to these issues is part of our future work.

As a possible criticism, it could be argued that building this type of models may be time-consuming and requires a medium-high level of expertise. This is why in section 2 we have stated that our method is not intended to be universal, it requires some conditions to be applicable: 1) the MS addressed is of general interest; 2) there are a lot of SP offered in this MS; 3) SP are big; 4) the selection activity is monitored by a team that accumulates experience from past selection processes; 5) the requirements are not absolutely known in advance. Although it seems to be a lot of restrictions, a great deal of selection processes in the information systems development satisfy them altogether (communication infrastructure, packages of various kinds: CRM, ERP, document management, ...); furthermore, the economical impact of the selection process in such cases is very high. We think that departments, institutes and teams continuously involved in selection activities, namely consultant companies (e.g., Gartner, Forrester, etc.), IT divisions in large organizations (including public ones) and groups of expert with academic profile, either large or smaller (e.g., our own team) are therefore the main targets of our proposal.

One could aim at formalizing in the future the decision of applicability of a MS or SP to a problem and convert our lightweight approach into heavyweight, but we must be aware that this requires lot of work to be done in the selection. First, the non-matching elements of the organizational model should be weighted somehow (as done for instance in [19] using AHP with all the involved stakeholders). Second, one should decide the threshold in which one MS become non-applicable. Third, some kind of decomposition of the goals is surely needed to further explore organizational needs and SP services. A lightweight approach is cost-effective and useful enough to have a first organizational analysis of the elements involved in selection; the utilization of a heavyweight one should be studied carefully.

For related work, we specially mention the notion of goal matching presented by Rolland [20] and the CARE approach by Chung and Cooper [21]. Both proposals are more comprehensive than ours in the sense that they propose to use goal models for driving the whole selection process, focusing on goal decomposition and creating goal graphs; whilst in our proposal the goal-oriented framework is to be used for a first approximation of the problem, with the goal of clarifying the high-level organizational model and pruning the solution space. In this sense, we may say that the late treatments proposed by Rolland and Chung are not contradictory but complementary to our proposal; in other words, they could be part of the heavyweight approach mentioned in the previous paragraph. For the rest of their proposals, the following fundamental differences arise:



- They compare directly organizational needs with product functionalities; no notion of MS model is proposed. This is a significant difference, in connection with the 2 fundamental questions stated in the introduction (proliferation and variety). We have already given arguments supporting the existence of this intermediate model.
- It is not clear how they tackle the usual situation in which a single SP does not solve the problem at hand. They mention the use of glue code and the further customization of the selected SP, but our experiences show that usually the information system must combine several SP. Our proposal deals with this situation in a natural way.
- Our formalization using UML is a good starting point for developing tool support in the future.

## Acknowledgements

This work has been partially supported by the CICYT programme, project TIN2004-07461-C02-01.

## References

1. J. Kontyo. "A Case Study in Applying a Systematic Method for COTS Selection". In *Proceedings 18<sup>th</sup> ICSE*, 1996.
2. N. Maiden, C. Ncube. "Acquiring Requirements for COTS Selection". *IEEE Software* 15(2), 1998.
3. C. Alves, F. Alencar, J. Castro. "Requirements Engineering for COTS Selection". In *Proceedings 3<sup>rd</sup> WER*, 2000.
4. S. Comella-Dorda, J. C. Dean, E. Morris, P. Oberndorf. "A Process for COTS Software Product Evaluation". In *Proceedings 1<sup>st</sup> ICCBSS*, LNCS 2255, 2002.
5. A. van Lamsweerde. "Goal-Oriented Requirements Engineering: A Guided Tour". In *Proceedings 5<sup>th</sup> ISRE*, 2001.
6. A. Dardenne, A. van Lamsweerde and S. Fickas. "Goal-Directed Requirements Acquisition". *Science of Computer Programming* Vol. 20, North Holland, 1993.
7. E. Yu, *Modelling Strategic Relationships for Process Reengineering*, PhD. thesis, University of Toronto, 1995.
8. GRL web page. <http://www.cs.toronto.edu/km/GRL/>. Last accessed November 2004.
9. A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, P. Traverso. "Specifying and analyzing early requirements in Tropos". *Requirements Engineering Journal*, 9(2), 2004.
10. E. Yu. "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering". In *Proceedings 3<sup>rd</sup> ISRE*, 1997.
11. Object Management Group. *UML 2.0*. <http://www.omg.org/>. Last accessed Nov. 2004.
12. X. Franch. "On the Lightweight Use of Goal-Oriented Models for Software Package Selection". Technical Report LSI-04-58-R, LSI-UPC, November 2004.
13. X. Burgués, C. Estay, X. Franch, J.A. Pastor, C. Quer. "Combined Selection of COTS Components". In *Proceedings 1<sup>st</sup> ICCBSS*, LNCS 2255, 2002.
14. J.P. Carvallo, X. Franch, C. Quer. "A Framework for Selecting Workflow Tools in the Context of Composite Information Systems". In *Procs. 15<sup>th</sup> DEXA*, LNCS 3180, 2004.

15. J.P. Carvallo, X. Franch, C. Quer. "Towards the Selection of a Requirements Management Tool". Book chapter in *Requirements Engineering for Sociotechnical Systems*, J.L. Maté, A. Silva (ed.), IDEA Group, 2005.
16. E. Yu, J. Mylopoulos. "Understanding "Why" in Software Process Modelling, Analysis, and Design". In *Proceedings 16<sup>th</sup> ICSE*, 1994.
17. A.I. Anton. "Goal-Based Requirement Analysis". In *Proceedings 2<sup>nd</sup> ICRE*, 1996.
18. McAfee web page. <http://www.mcafee.com>. Last accessed November 2004.
19. H. Kaiya, H. Horai, M. Saeki. "AGORA: Attributed Goal-Oriented Requirements Analysis Method". In *Proceedings 10<sup>th</sup> RE*, 2002.
20. C. Rolland. "Requirements Engineering for COTS Based Systems". *Information and Software Technology*, 41, Elsevier, 1999.
21. L. Chung, K. Cooper. "Defining Goals in a COTS-Aware Requirements Engineering Approach". *System Engineering Journal*, 7(1), 2004.

# Measuring IT Infrastructure Project Size: Infrastructure Effort Points

Joost Schalken<sup>1</sup>, Sjaak Brinkkemper<sup>2</sup>, and Hans van Vliet<sup>1</sup>

<sup>1</sup> Vrije Universiteit, Department of Computer Science,  
De Boelelaan 1083a, 1081 HV Amsterdam, The Netherlands  
{j.j.p.schalken, j.c.van.vliet}@few.vu.nl

<sup>2</sup> Utrecht University, Institute of Information and Computing Sciences,  
Padualaan 14, 3584 CH Utrecht, The Netherlands  
{s.brinkkemper}@cs.uu.nl

**Abstract.** Our objective is to design a metric that can be used to measure the size of projects that install and configure COTS stand-alone software, firmware and hardware components. We call these IT infrastructure, as these components often form the foundation of the information system that is built on top of it. At the moment no accepted size metric exists for the installation and configuration of stand-alone software, firmware and hardware components. The proposed metric promises to be a viable instrument to assess the effectiveness and efficiency of IT infrastructure projects.

## 1 Introduction

Organizations no longer create software intensive systems from scratch. The use of pre-existing software components, not created by the organizations themselves, becomes ever more prevalent in the creation of large software systems [1, 2]. These pre-existing components are often called *off-the-shelf components* (COTS components) in software engineering literature. In this paper we however prefer to use the term *non-developmental software components* (NDIs) [3] for these pre-existing components.

The integration of NDI components that are packaged as stand-alone programs differs significantly from traditional software development. Many software engineering metrics that have been applied to software development (such as function points [4, 5], lines of code [6], object points [7], or bang metrics [8]) cannot be applied to projects that integrate these NDI components into larger systems. After all most effort in infrastructural IT projects is not in programming a system, but in installing and configuring the system. In this paper we propose a new software metric to measure the size of projects that integrate stand-alone NDI components into software-intensive systems.

The metric we propose is not only applicable to the integration of stand-alone non-developmental software components, but also to the integration of

firmware<sup>1</sup> and hardware components into software-intensive systems. We use the term *IT infrastructure* (IT infrastructure) to refer to NDI hardware, firmware and stand-alone software components, as these components often form the foundation of the information system that is built on top of it.

Examples of IT infrastructure projects are: operating system upgrades, installations of a database system, deployment of new desktop computers and memory upgrades of servers.

Up until now no size metric in the area of IT infrastructure development has received broad acceptance by industry. Although no standard size metric for IT infrastructure exists, it does not mean that there is no need for such a metric. On the contrary, the need for such a size metric is increasing. Empirical knowledge of NDI-based systems is still at an early stage of maturity [9]. Processes are quite different from traditional projects [10] and project estimation and tracking are less effective for NDI-based development [10]. This is problematic as the use of NDI components is becoming ever more prevalent [1, 2]. The metric we propose in this paper uses information part of which only becomes available late in a project. Consequently, it's intended use is to assess and validate the effectiveness and efficiency of projects, rather than upfront cost estimation.

## 1.1 IT Infrastructure Defined

The absence of consensus on the meaning of COTS within the academic community [3, 11] necessitates a definition of the related concept IT infrastructure in this section. The definition of IT infrastructure marks which projects can and which projects cannot be measured with the new size metric.

The term IT infrastructure has been inspired by the term technical infrastructure [12]. In this paper the following definition of *IT infrastructure* will be used: “*IT infrastructure is the hardware, software, and services that support the operation of an information system.*”

The development of IT infrastructure is concerned with pre-existing hardware and software components that have not been developed by the organizational unit that installs these components. In software engineering literature a pre-existing software component is often called a *commercial off-the-shelf component* (COTS component). In this paper we however prefer to use the term *non-developmental component* (NDI) [3], as the term COTS implies that the component comes from a commercial vendor. In this paper the term NDI refers not only to software components, but also to firmware and hardware components.

Non-developmental software components can be packaged in several ways [11], either as source code, static libraries, dynamic libraries, binary components or stand-alone programs. The type of packaging also has a direct impact on how

---

<sup>1</sup> As firmware has remarkable similarity with software (with respect to its configuration), everywhere where software is written, one should read software/firmware unless explicitly stated otherwise.

these components are integrated and delivered to the customer. NDI components that are provided in source code or library form usually require programming to integrate the components into the software system under construction. These kinds of NDI components are usually delivered as inseparable subcomponents of the system under construction. On the other hand NDI components that are provided as stand-alone programs usually require little or no programming, but require so much the more configuration. These NDI components are usually not delivered as an inseparable system, but instead the components need to be installed separately or are installed by the installation program as separate, stand-alone components or programs.

The development of IT infrastructure not only entails the installation and configuration of stand-alone software components, but also the deployment, connection and configuration of hardware components. As the choice of software components is not independent hardware components and because the integration of the components is frequently performed in a single project, we have chosen to measure the size of software, firmware and hardware components using a single, composite size metric.

The development of IT infrastructure consists of *hardware* (placing the physical components), *software* (loading the software from the installation medium into the target hardware), *hardware* (installing sub-modules and wiring the hardware) and *software* (setting and testing the configurable parameters and settings).

## 1.2 Related Work

This section describes related work in the field of IT infrastructure size metrics and algorithmic cost estimation models for IT infrastructure. Cost estimation models are included in the overview, as they can be seen as size models that do not only take the inherent problem size into account, but also the capability of the environment to deal with the complexity at hand. In this article we focus solely on the costs of labor for the installation and configuration of the IT infrastructure. Selection effort, hardware costs, such as discussed in [13], and license costs lie outside the scope of this paper.

There are two cost estimation methods that are commonly used in practice: *service point analysis* and *service level agreement analysis*. The IT infrastructure component is either viewed as a service whose costs are of a recurring nature or as a component that is delivered to an organization as product or a one-off service delivery. Examples of IT infrastructure services are cpu cycles on a mainframe and network ports. Examples of IT infrastructure products are configured servers and software upgrades.

The most crude approach to IT infrastructure costing consists of amortizing the total costs generated by a class of IT services or IT products by total amount of service units or products delivered.

A more sophisticated approach to IT infrastructure service costing is offered by Van Gorkom [14] in the form of Service Point Analysis. The Service Level Agreement is decomposed into Service Level Agreement components. Based on

the Service Level Agreement components standardized cost estimates can be made.

Another more sophisticated approach to IT infrastructure product sizing is the SYSPPOINT method [15]. The IT infrastructure product to be provided is divided into primitive components (servers, workstations, printers, LAN's, WAN's, server applications and client applications). Based on the relative complexity and count of the primitive components, tables can be used to calculate the total size of a project in SYSPPOINTS.

COCOTS [16, 17] is an advanced cost-estimation model for NDI software, based on the COCOMO suite of cost estimation models. The COCOTS model allows the estimation of not only the implementation of the system, but also the selection costs and modification costs. The COCOTS tailoring model estimates the implementation and configuration costs of a system, based on parameter specification, script writing, reports & gui, security and the availability of tailoring tools. Each of the factors is measured on a five-point scale.

The last method discussed in this section is data envelopment analysis, which can be applied to measure the relative efficiency in creating IT infrastructure [12]. Data envelopment analysis allows the efficiency of projects to be compared on a variety of output measures simultaneously. In that sense it is not a cost estimation or size measurement procedure, but the method does shine a light on the project's productivity. The method solves the problem that IT infrastructure can have multiple outputs (e.g. servers can have connected users, storage space and processing speeds as output measures).

Amortizing product or service costs, Service Point Analysis and the SYSPPOINT method share the drawback that estimates can only be made for IT infrastructural systems that consist of known infrastructural product or service components. The costs for IT infrastructural systems that contain novel product components or service components cannot be estimated.

Data envelopment analysis can only analyze the implementation efficiency of projects that implement similar products or services. Different IT infrastructural projects will yield very different primitive outputs. Compare the efficiency in providing network throughput with the efficiency of providing storage space (both measured in gigabytes). This explains why comparisons are only possible between projects that have similar end results, comparing throughput with storage of data is of course not meaningful.

Our method measures the size of IT infrastructure on a continuous, interval scale. It is reasonably precise, whereas COCOTS measures each attribute on a rough 5-point scale. The metric allows different types of IT infrastructural products to be compared to each other and does not depend on the existence of a list of known IT infrastructure components.

### 1.3 Structure of Paper

Having discussed the necessity of IT infrastructure metrics and the definition of IT infrastructure, the remainder of this paper is structured as follows: In Sect. 2 the metaphor that guided the design of the infrastructure metric is presented

together with the formal definition of the metric. Section 3 discusses the calibration of the measurement formulas presented in Sect. 2. Section 4 provides the results of the preliminary calibration and validation of the metric during the feasibility study. Section 5 describes the conceptual validation of the proposed metric for IT infrastructure. And the last section wraps up the paper with some concluding remarks and further work.

## 2 Infrastructure Effort Points

In this section we present our metric to measure the size of IT infrastructure projects, the Infrastructure Effort Points, or IEP for short. First the the underlying principles that guided the design of the size metric are explained. Following the theory design of the metric a detailed description of Infrastructure Effort Points and its measurement procedure is given.

Infrastructure Effort Points only measure the size of the installation and configuration effort of the IT infrastructure. The NDI component selection effort, training effort, hardware costs, and license costs cannot be measured using this metric.

### 2.1 Theory Design

In this section we explain the design of the size metric for IT infrastructure using a metaphor. A metaphor helps to create a common understanding [18]. It can be used to explain the underlying principles and assumptions of a measurement procedure.

The metaphor gives an intuitive justification for a metric. For example function point analysis counts the variety and complexity of the data records that are received, stored, transmitted and presented by an information system. Function point analysis is able to measure the size of different classes of information systems by abstracting each information system to a manipulator of flows of information. This is the guiding metaphor of function point analysis. It is based on the assumption that the complexity of an information system is equal to the complexity of its information flows.

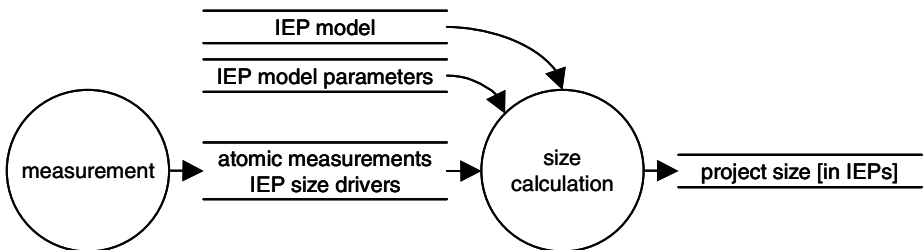


Fig. 1. Calculation of size based on project characteristics

**Table 1.** Atomic measurements for IEP of hardware installation

<i>Group</i>	<i>Size driver</i>	<i>Symbol</i>	<i>Unit of measurement</i>
main components	number of components	$c_i$	-
	average weight	$c_i^w$	kilo
	installation or removal	$c_i^a$	{installed, removed}
subcomponents	number of subcomponents	$s_{i,j}$	-
	average number of connections	$s_{i,j}^c$	-
	installation or removal	$s_{i,j}^a$	{installed, removed}
external connections	number of connections	$w_{i,j}$	-
	average length	$w_{i,j}^l$	meter
	installation or removal	$w_{i,j}^a$	{installed, removed}

Infrastructure Effort Point analysis considers the development of IT infrastructure to consist of two activities: wiring and placing hardware boxes during the deployment and connection of hardware components and the manipulation of configuration settings during the configuration of software components and configurable hardware.

The guiding metaphor for Infrastructure Effort Points is based on the following three assumptions:

1. Infrastructural IT projects are composed of a hardware and a software component.
2. The effort of the hardware component of the project depends on the number of hardware boxes that need to be installed and the number of connections that need to be made.
3. The effort of the software component of the project depends on the number of configuration parameters that need to be set.

## 2.2 Hardware Effort Points

Two distinct tasks in IT infrastructure projects have been identified: tasks related to hardware and tasks related to software. Verner and Tate [19] argue that different types of system components can have different size equations. In this section we identify the size drivers and size equations that are applicable to the deployment and connection of hardware components.

A bottom-up size model, as the Infrastructure Effort Points, consists of a number of size drivers and one or more size equations. A size driver is “any countable, measurable, or assessable element, construct, or factor thought to be related to the size” of a component [19]. The size drivers form the input to a size equation that combines the different counts on a specific size driver into a single size measurement.

For the hardware side of the Infrastructure Effort Point equation we distinguish three major size drivers: main components, subcomponents and connections. Each of the major size drivers has associated minor size drivers that can be used to fine-tune the size equations in the future. Although technically there is no difference between a major and a minor size driver, practically we expect the



major size drivers to have greater influence on the size of the IT infrastructure project.

The first major size driver is the number of main components  $c_i$  of a certain type of hardware that has been installed or removed. Main components are those pieces of hardware that are considered to form a functional whole by their average end users. E.g. an end user will identify a scanner as a main component, whereas the separate automatic document feeder for the scanner is seen as a subcomponent, as the automatic document feeder cannot be seen as an independent machine that offers functionality on its own. Associated minor size drivers are the average weight of the main component  $c_i^w$  and whether the components were installed or removed  $c_i^a$ .

The second major size driver is the number of subcomponents  $s_{i,j}$  of a certain type that have been installed or removed from main component  $c_i$ . (The index  $i$  refers to the main component to which the subcomponents are attached, the index  $j$  refers to this particular group of subcomponents.) Subcomponents that have not been installed or removed, but instead were already assembled with the main component should not be counted. Minor size drivers that are associated with the size driver number of subcomponents are the average number of connections between the subcomponent and the main component and other subcomponents  $s_{i,j}^c$  and whether the subcomponents were installed or removed  $s_{i,j}^a$ .

The third and last major size driver for the hardware side is the number of connections  $w_{i,j}$  between the outside world and main component  $c_i$ . The connections size driver considers all physical connections (wires) between the main component and the outside world, but not the mutual connections between subcomponents and connections between subcomponents and the main component as these have already been counted (in  $s_{i,j}^c$ ). Examples of connections are the power cable and the network cable of a personal computer, but not the keyboard cord. Associated minor size drivers are the average length of the connection  $w_{i,j}^l$  and whether the connections were installed or removed  $w_{i,j}^a$ .

These three major size drivers and their associated minor size drivers form the input for the size equation that combines the measurements on the individual size drivers, see Fig. 1 for a schematic overview. The size equations consist of a model combined with calibrated model parameters. The equation model states which size drivers need to be taken into account and in which manner. E.g. a size model might state that the size of a task is equal to the number of main hardware components multiplied by a constant plus the number of connections multiplied by a constant, i.e. size  $s_i^{hw} = \theta_1 \cdot c_i + \theta_2 \cdot w_{i,j}$ . The exact size equation is determined during the calibration process in which the most appropriate values for the constants in the equation are determined using empirical data.

The calibration of a size model, once the empirical data has been collected, is conceptually easy, albeit computer-intensive (more information on the calibration process can be found in Sect. 3). The selection of the appropriate model is more complicated. Apart from the selection of an appropriate form for the equation (linear, logarithmic or quadratic) one needs to be careful to select the

**Table 2.** Atomic measurements for IEP of software configuration

<i>Attribute</i>	<i>Metric</i>	<i>Symbol</i>	<i>Unit of measurement</i>
configuration parameters	number of parameters	$p_i$	-
	parameter type	$p_i^v$	{text, number, boolean, binary}
	input type	$p_i^t$	{gui, text interface, configuration file, configuration database, script file, dip-switch/jumper, other}
configuration group	number of parameters	$g_{i,j}$	-
	parameter type	$g_{i,j}^v$	<i>see above.</i>
	input type	$g_{i,j}^t$	<i>see above.</i>

right amount of size drivers for the model. Too few size drivers makes the size equation perform poorly, as insufficient information is taken into account. Too many size drivers creates the risk of over-fitting the size model, causing the size equation not to capture the real hardware size but instead some random patterns that exist in the observed data. The risk of over-fitting the model is increased when many size drivers are included relative to the amount of empirical data.

For the first empirical validation we propose to use only a single size driver to prevent over-fitting the data. When more data becomes available more complex size models can be examined. The most important hardware size driver is the number of main components. We therefore propose to use the following simple formula to calculate the size of the hardware part  $s^{hw}$ .

$$s^{hw} = \sum_{i=1} \theta_1^{hw} \cdot c_i \quad (1)$$

### 2.3 Software Effort Points

For the hardware side of the Infrastructure Effort Point equation we should be able to apply the method to all possible situations, for the software side of the Infrastructure Effort Point equation we differentiate between two usage scenarios. In the first scenario the software is configured for the first or second time in the organisation (configuration engineering), whereas in the second scenario the organisation has a degree of experience in configuring the software (configuration roll-out). In a large deployment of software within an organisation one usually starts with configuration engineering and when all the required information about configuring the software has been gathered the project proceeds with configuration roll-out.

The difference between a first-time configuration and a repeat configuration is the familiarity with the software product. During a first installation one needs to examine all configuration parameters to determine which parameters require adjustment. When one is familiar with a system one knows which parameters require adjustment and which factory settings are already correct.

For the software side of the Infrastructure Effort Point equation we distinguish one or two major size drivers. With configuration roll-out projects the major size driver is the number of configuration parameters. In recognising the effort required to determine which parameters to change, configuration-engineering has a second major size driver the total amount parameters in a group of parameter settings, that measures all available configuration parameters.

The major size driver for software configuration tasks is the number of configuration parameters  $p_i$  that require modification. During the installation of a software component the software is loaded from the installation medium to the target execution platform and simultaneously the settings of the components are set. The type of settings that determine the behaviour of a component can vary, broadly from installation directories, subsystem selection, user account creation, user settings to script files. Associated minor size drivers are: the type of values a parameter can store  $p_{i,j}^v$  and the input method that is required to change the parameter value  $p_{i,j}^t$ . The type of parameter values has an influence on the effect size of the task, because binary strings are much harder to enter and test compared to boolean parameters. The input method also has influence on the configuration size, as for example configuration using a gui is easier than configuring a system using a configuration file.

The other major size driver for configuration engineering tasks is the number of configuration parameters in the configuration group  $g_{i,j}$  that belong to the configuration parameter  $p_i$ . As configuring a system with only a few parameters is easier as configuring a system with a large number of parameters, the number of available configuration parameters also needs to be taken into account into the size equation. Therefore, all existing configuration parameters, that are seen by the IT team during installation and configuration are counted. The associated minor size drivers are the same as those described for the size driver number of configuration parameters.

To calculate the size for roll-out configuration tasks, the following model formula can be used:

$$s^{sw} = \sum_{i=1} \theta_1^{sw} \cdot p_i \tag{2}$$

To calculate the size for configuration-engineering tasks, the following model formula can be used:

$$s^{sw} = \sum_{i=1} (\theta_1^{sw} \cdot p_i) + \sum_{j=a} (\theta_2^{sw} \cdot g_{i,j}) \tag{3}$$

## 2.4 Infrastructure Effort Points

Having explained both the hardware and software part of the Infrastructure Effort Point measurement, we are ready to combine these two measurements into the overall Infrastructure Effort Point measurement.

To obtain the (total) Infrastructure Effort Point measurement the Hardware Effort Points and the Software Effort Points are added. To prevent one of the

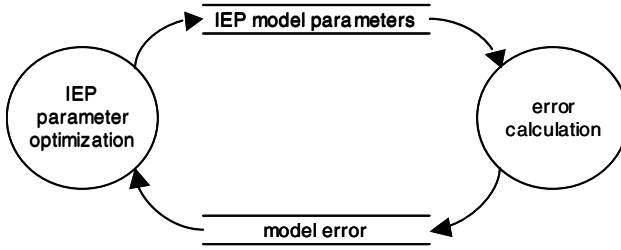


Fig. 2. Optimization of the IEP model parameters

two measurements overruling the other measurement, a scaling factor  $\theta^{scale}$  is used. This leads to the following equation:

$$s^{hw} = s^{hw} + \theta^{scale} \cdot s^{sw} \quad (4)$$

### 3 Calibration Process

The purpose of the calibration is to find the IEP model parameters for an IEP size model. Together with the size model, the IEP model parameters make up the estimation equation of the Infrastructure Effort Points.

A good size metric should correlate well with development effort. We therefore define an optimal set of IEP model parameters to be those parameters that allow a cost estimation model to make the best possible prediction/explanation of the costs of a project. The advantage of using a cost estimation model as an intermediate to correlate size with development effort are twofold. First the cost estimation model can take phenomena as diseconomy of scale and schedule compression into account. The second advantage is that once the model has been calibrated using empirical data, other people can more easily recalibrate the model to their specific environment; the only need to recalibrate the estimation parameters.

To find the optimal calibration of the measurement model, empirical data about the infrastructural size drivers and the development effort is needed. The optimization process consists of the cyclic process of improving/modifying the initial IEP model parameters and consequently calculating error of the estimation algorithm (compared with the historical performance), see Fig. 2.

### 4 Feasibility Study

Having described the definition of the Infrastructure Effort Points and its calibration process, we describe the results of a preliminary feasibility study conducted to test the practicality of the data collection and its associated calibration process.

**Table 3.** Aggregated data from feasibility study

<i>project name</i>	<i>effort</i>	<i>components</i>	<i>configuration</i>	<i>configuration</i>
			<i>roll-out</i>	<i>engineering</i>
			<i>parameters</i>	<i>parameters</i>
			<i>changed</i>	<i>changed</i>
				<i>total</i>
Easy Icon Maker	7 min		0	1
Internet Explorer 5.5	10 min		13	20
MS Office 200 Professional	35 min		11	169
MS Virtual PC 2004	20 min		12	29
MS Windows ME	113 min		29	375
Printer installation	30 min	1		
Staff computer installation	120 min	1		
TCP/IP setup Windows ME	12 min			6
User info in MS Word 2000	9 min		5	54

To test the data collection and calibration process, we collected measurements and effort data of nine projects that were collected in a controlled environment. The projects consisted of both hardware and software infrastructure projects. The aggregated results can be seen in Table 3<sup>2</sup>.

The conclusion of the feasibility study is that it is possible to collect the required data in an efficient manner that does not disrupt the IT infrastructure project itself. All that is required is some degree of discipline in recording the steps during the project. Based on the data we are able to obtain a preliminary calibration of the data, however as this calibration is based only on nine projects it is not statistically significant.

## 5 Measurement Validation

Software metrics need to be validated to ensure that they measure what they purport to measure [20]. The validation of the metric should check whether the metric is valid and/or correct.

The validity of a software metric refers to its ability to provide measurements that can be used by practitioners in the context in which the metric was gathered. The correctness of a software metric refers to generic “laws” that govern measurements in general. An example of a correctness requirement is that addition of two measurement values should lead to a meaningful total. This requirement and other many correctness requirements are codified in measurement theory (see e.g. .

As correctness requirements on software metrics have been extensively discussed in the academic literature (e.g. [21, chap. 3]) they will not be discussed in more detail in this paper. In following section we pay more attention to the validity of the Infrastructure Effort Point metric. The validity requirements that are discussed in this section are: the focus aspect, the objectivity aspect, the timeliness aspect, the granularity aspect, and the genericity aspect.

<sup>2</sup> The full dataset of the preliminary data collection is available in Microsoft Access-form, from the following address: <http://www.cs.vu.nl/reflection/infra-metrics/>.

## 5.1 Validity Requirements

The *work focus* of a software size metric determines whether the emphasis of the size of an IT solution lies on the size of the problem to be solved with an IT solution or on the size of the IT to create the solution. Certain complex function requirements on a system might take little implementation effort given the right tools whereas certain apparently simple functional problems might require a large amount of implementation effort because tool support is lacking. Metrics with a *problem focus* pay more attention to the size of the problem and less on the size of the IT required to solve the problem. Metrics with a *work focus* pay more attention to the size of the IT solution as compared to the problem.

The IT infrastructure size metric has a work focus, as typical infrastructural IT projects are not free to choose which infrastructure to implement to support the functional requirements of the user. Lacking influence on the choice and availability of suitable IT infrastructure, it would not be fair to hold projects accountable for their implementation efficiency.

The *objectivity* of a software size metric dictates whether the determination of the size of a software can be based partly on human judgment or can only be based on rules that can be interpreted in only a single manner. *Objective* metrics require only the application of clear rules and require no human judgment. *Subjective* metrics on the other hand do require human judgment and interpretation before a size can be determined. C.f. lines of codes [6] which can be counted automatically by a computer (a fully objective metric) with function points [5] (a partially subjective metric) which require a human function point counter to interpret the requirements specification. Size metrics should be as objective as possible, as subjectivity leaves room for disagreements on the real functional size and causes inaccuracies (e.g. [22]). The objectivity aspect of a metric is not a crisp distinction between objective and subjective, but a wide spectrum of nuances is possible.

The IT infrastructure size metric is an objective measure of the size of an IT infrastructural task. The determination of which hardware components are main components and which are sub-components does involve some subjective judgement. The same holds for the determination of the number of parameters in a parameter group, as the boundaries of the parameter group are not always very clear.

The *availability* of a software size metric determines in which phase of the project one needs to be able to determine the size. Size measurements require information about a project or piece of software that becomes available in the course of the project. Certain information is available already at an early stage of a project whereas other information only becomes available near the end of the project.

The IT infrastructure size metric is meant to be used for assessment and evaluation of methods and practices. This usually takes place at the end of a project, in contrast to practices such as estimation and project tracking that take place at an earlier phase of the project. As assessment and evaluation take

place at the end of a project, it is not problematic if some of the required data only becomes available near the end of a project.

The *granularity* of a software size metric refers to the aggregation level to which the metric is applied. The efficiency can be determined of a single project, of all work related to a single product type or of all work in an organizational unit. E.g. the efficiency of single network installation project can be measured (e.g.  $\frac{\text{cost}}{\text{number of network points}}$ ) or the efficiency of all networking operations in an organization based on cost per network point can be measured (e.g.  $\frac{\text{cost}}{\text{number of network points}}$ ). Between project-level and organizational-level metrics lie the *product-level* metrics that measure the efficiency of implementing a single type of product (e.g. a Linux server).

The IT infrastructure size metric needs to have a project-level granularity to explain in which contexts methods do work and in which contexts they do not work. Metrics with an organizational-level granularity would obscure the reasons why certain processes do or do not work.

The *applicability* of a software size metric indicates how broad the applicability of the metric should be. *Product-level* metrics can be applied to a broad range of software products, whereas *project-level* metrics can only be applied to a limited range of software products.

The IT infrastructure size metric needs to be a general-purpose metric. IT infrastructure includes a broad area of products. One can only meaningfully compare efficiency rates that are based on the same size measurements. It would therefore be beneficial if most IT infrastructure can be measured with the metric, allowing comparisons of the applicability of processes in different infrastructural domains.

## 6 Conclusions

In this paper we discuss the design and principles of the Infrastructure Effort Point metric for IT infrastructure projects. The metric is an objective metric that can be used to measure the size of IT infrastructure projects. It outperforms other existing size metrics for IT infrastructure in genericity and objectivity.

Infrastructure Effort Points can help to assess the effectiveness of processes and techniques, making it a valuable tool for process improvement for organizations that deliver IT infrastructure.

The Infrastructure Effort Point metric is unsuitable for the upfront estimation of a project's schedule or costs. The required information to feed into the size equation is available only at a late stage of the project. However with the aid of additional techniques (e.g. PROBE [23, pp. 109–134]) it might well be possible to fruitfully use the metric for estimation as well.

The results look very promising, but some work still needs to be done. First, data about IT infrastructural project containing at least the IEP size drivers and the effort consumed by the project needs to be collected. This collected database will be used to calibrate the parameters in the Infrastructure Effort

Point model and to analyse how good Infrastructure Effort Points correlate with the real expended effort.

## References

1. Abts, C., Boehm, B.W., Clark, E.B.: Observations on COTS software integration effort based on the COCOTS calibration database. In: Proceedings of the Twenty-Fifth Annual Software Engineering Workshop (SEW 25), NASA/Goddard Space Flight Center (2000) Available from: [http://sel.gsfc.nasa.gov/website/sew/2000/topics/CABts\\_SEW25\\_Paper.PDF](http://sel.gsfc.nasa.gov/website/sew/2000/topics/CABts_SEW25_Paper.PDF).
2. Morisio, M., Seaman, C.B., Basili, V.R., Parra, A.T., Kraft, S.E., Condon, S.E.: COTS-based software development: Processes and open issues. *Journal of Systems and Software* **61** (2002) 189–199
3. Carney, D., Long, F.: What do you mean by COTS? finally, a useful answer. *IEEE Software* **20** (2000) 83–86
4. Albrecht, A.J.: Measuring application development productivity. In: Proceedings of the Joint SHARE/GUIDE/IBM Applications Development Symposium. (1979) 83–92
5. Albrecht, A.J., Gaffney, Jr., J.E.: Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering* **9** (1983) 639–648
6. Park, R.E.: Software size measurement: A framework for counting source statements. Technical Report CMU/SEI-92-TR-020, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA (1992) Available from: <http://www.sei.cmu.edu/>.
7. Banker, R.D., Kauffman, R.J., Wright, C., Zweig, D.: Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment. *IEEE Transactions on Software Engineering* **20** (1994) 169–187
8. DeMarco, T.: *Controlling Software Projects: Management, Measurement & Estimation*. Yourdon Computing Series. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA (1982)
9. Basili, V.R., Boehm, B.: COTS-based systems top 10 list. *Computer* **34** (2001) 91–93
10. Morisio, M., Seaman, C.B., Parra, A.T., Basili, V.R., Condon, S.E., Kraft, S.E.: Investigating and improving a COTS-based software development process. In: Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, I, IEEE Computer Society Press (2000) 32–41
11. Morisio, M., Torchiano, M.: Definition and classification of COTS: a proposal. In Dean, J., Gravel, A., eds.: Proceedings of the 1st International Conference on COTS Based Software Systems (ICCBBS 2002). Volume 2255 of Lecture Notes in Computer Science., Berlin, D, Springer-Verlag (2002) 165–175
12. Stensrud, E., Myrtveit, I.: Identifying high performance ERP projects. *IEEE Transactions on Software Engineering* **29** (2003) 398–416
13. Ardagna, D., Francalanci, C., Trubian, M.: A cost-oriented approach for infrastructural design. In: Proceedings of the 2004 ACM symposium on Applied computing, ACM Press (2004) 1431–1437
14. van Gorkom, V.: Service Point Analysis: A study on its effectiveness and usefulness. M.sc. thesis, CIBIT, Utrecht, NL (2002)



15. Raghavan, S., Achanta, V.: SYSPPOINT: Unit of measure for IT infrastructure project sizing. *Journal of Computing and Information Technology* **12** (2004) 31–46
16. Abts, C.M., Boehm, B.W.: COTS software integration cost modeling study. Technical report, Center for Software Engineering, University of Southern California, Los Angeles, CA, USA (1997)
17. Boehm, B.W., Abts, C.M., Bailey, B.: COCOTS software integration cost model: Insights and status (1999) Presented at: Ground System Architectures Workshop (GSAW-99).
18. Robinson, H., Sharp, H.: XP culture: Why the twelve practices both are and are not the most significant thing. In: *Proceedings of Agile Development Conference*, Washington, DC, USA, IEEE Computer Society Press (2003) 12–21
19. Verner, J., Tate, G.: A software size model. *IEEE Transactions on Software Engineering* **18** (1992) 265–278
20. Schneidewind, N.F.: Methodology for validation software metrics. *IEEE Transactions on Software Engineering* **18** (1992) 410–422
21. Fenton, N.E., Pfleeger, S.L.: *Software Metrics: A Rigorous and Practical Approach*. 2nd edn. International Thomson Computer Press, London, UK (1998)
22. Kemerer, C.F.: Reliability of function points measurement: a field experiment. *Communications of the ACM* **36** (1993) 85–97
23. Humphrey, W.S.: *A Discipline for Software Engineering*. Addison-Wesley Professional, Boston, MA, USA (1994)

# Author Index

- Akkermans, Hans 400  
Andersson, Birger 233  
Araújo, João 293
- Baida, Ziv 400  
Benatallah, Boualem 415  
Benavides, David 491  
Bergholtz, Maria 233  
Bertino, Elisa 119  
Bézivin, Jean 309  
Brinkkemper, Sjaak 567
- Cabibbo, Luca 135  
Cabot, Jordi 48  
Carosi, Antonio 135  
Casati, Fabio 415  
Casteleyn, Sven 63  
Chen, An-Pin 387  
Chen, Mu-Yen 387  
Cuadra, Dolores 119
- da Silva, Roberto 187  
Dalamagas, Theodore 201  
De Troyer, Olga 63  
Di Ruscio, Davide 475  
Dittrich, Klaus R. 105  
dos Santos Mello, Ronaldo 151  
do Prado Leite, Julio Cesar Sampaio 535
- Eder, Johann 248  
Edirisuriya, Ananda 233  
Edmond, David 216  
Etien, Anne 277  
Evermann, Joerg 33
- Franch, Xavier 551  
Freytag, Johann-Christoph 167
- Gordijn, Jaap 400  
Gordillo, Silvia 446  
Grégoire, Bertrand 430  
Grigori, Daneila 415  
Guelfi, Nicolas 430
- Hacmac, Roger 105  
Hammoudi, Slimane 309  
Heuser, Carlos Alberto 151, 187  
Hodel, Thomas B. 105  
Holze, Marc 90  
Hoorn, Johan F. 357  
Hoppenbrouwers, S.J.B.A. 262
- Ilayperuma, Tharaka 233
- Jablonski, Stefan 90  
Johannesson, Paul 233  
Jouault, Frédéric 309
- Kamsties, Erik 519  
Karagiannis, Dimitris 77  
Kittivoravitkul, Sasivimol 460  
Kühn, Harald 77
- Lehmann, Marek 248  
Liu, Lin 535  
Lopes, Denivaldo 309
- Mammar, Amel 430  
Martínez, Paloma 119  
McBrien, Peter 326, 460  
Morch, Andrei Z. 400  
Moreira, Ana 293  
Muñoz, Javier 342  
Mylopoulos, John 535
- Necib, Chokri Ben 167  
Neto, Pedro Santos 504  
Nezhad, Hamid R. Motahari 415
- Olivé, Antoni 1
- Pádua, Clarindo 504  
Pelechano, Vicente 342  
Petrov, Ilia 90  
Pierantonio, Alfonso 475  
Plessers, Peter 63  
Pohl, Klaus 519  
Proper, H.A. 262

- Ram, Sudha 32  
Ramel, Sophie 430  
Rashid, Awais 293  
Reis, Sacha 519  
Resende, Rodolfo 504  
Reuys, Andreas 519  
Riggio, Roberto 77  
Rizopoulos, Nikolaos 326  
Rolland, Colette 277  
Rossi, Gustavo 446  
Ruiz-Cortés, Antonio 491  
Russell, Nick 216
- Sæle, Hanne 400  
Schalken, Joost 567  
Schmitt, Michael 430  
Schwabe, Daniel 446  
Stasiu, Raquel Kolitski 187
- Teniente, Ernest 48  
ter Hofstede, Arthur H.M. 216  
Theodoratos, Dimitri 201  
Toumani, Farouk 415  
Trinidad, Pablo 491, 535
- Ursino, Domenico 77
- van der Aalst, Wil M.P. 216, 372  
van der Raadt, Bas 357  
van der Weide, Th.P. 262  
van Dongen, B.F. 372  
van Vliet, Hans 357, 567  
Vanderdonckt, Jean 16  
Verbeek, H.M.W. 372
- Yu, Eric S.K. 535  
Yu, Yijun 535