BIRKHÄUSER

Whitestein Series
in Software Agent Technologies
and Autonomic Computing

Emerging Web Services Technology

Volume II

Thomas Gschwind
Cesare Pautasso
Editors

BIRKHÄUSER

Whitestein Series in Software Agent Technologies and Autonomic Computing

The Whitestein Series in Software Agent Technologies and Autonomic Computing reports new developments in agent-based software technologies and agent-oriented software engineering methodologies, with particular emphasis on applications in the area of autonomic computing & communications.

The spectrum of the series includes research monographs, high quality notes resulting from research and industrial projects, outstanding Ph.D. theses, and the proceedings of carefully selected conferences. The series is targeted at promoting advanced research and facilitating know-how transfer to industrial use.

About Whitestein Technologies

Whitestein Technologies is a leading innovator in the area of software agent technologies and autonomic computing & communications. Whitestein Technologies' offering includes advanced products, solutions, and services for various applications and industries, as well as a comprehensive middleware for the development and operation of autonomous, self-managing, and self-organizing systems and networks.
Whitestein Technologies' customers and partners include innovative global enterprises, service providers, and system integrators, as well as universities, technology labs, and other research institutions.

www.whitestein.com

Emerging Web Services Technology

Volume II

Thomas Gschwind
Cesare Pautasso
Editors

Birkhäuser
Basel · Boston · Berlin

Editors:

Thomas Gschwind
Research Laboratory
IBM Research Division GmbH Zürich
Säumerstrasse 4
8803 Rüschlikon
Switzerland
e-mail: thg@zurich.ibm.com

Cesare Pautasso
Faculty of Informatics
University of Lugano
Via G. Buffi 13
6904 Lugano
e-mail: c.pautasso@ieee.org

2000 Mathematical Subject Classification: 68-06, 68Q60, 68T05, 68T27, 68T35, 68U35

Library of Congress Control Number: 2007929515

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

ISBN 978-3-7643-8863-8 Birkhäuser Verlag AG, Basel – Boston – Berlin

ISBN 978-3-7643-8863-8 e-ISBN 978-3-7643-8864-5

9 8 7 6 5 4 3 2 1 www.birkhauser.ch

Contents

Preface

The 2^{nd} Workshop on Emerging Web Services Technology (WEWST'07) was colocated with the 5^{th} European Conference on Web Services (ECOWS'07) which took place in November 2007 in Halle (Saale), Germany.

WEWST focuses on research contributions advancing the state of the art in Web Services technologies. The main goal of the WEWST workshop is to serve as a forum for providing early exposure and much needed feedback to grow and establish original and emerging ideas within the Web Services community. The wide variety of tools, techniques and technological solutions presented in WEWST share one common feature: they advance the current Web Services research in new directions by introducing new, sometimes controversial, ideas into the field. As such, WEWST is the natural extension to the main ECOWS conference.

As it can be seen from the workshop program, the spectrum of research topics related to such emergent technologies includes: the challenge of adopting RESTful Web Services and Resource Oriented Architectures; Dynamic Web Service Discovery, Selection and Composition; extensions to the standard Business Process Execution Language; the management of composite Web Services; the delivery of well defined Quality of Service guarantees; the performance evaluation of Web Services. These are all still among the hot topics in Web Services research since no satisfactory solution has been found yet.

We would like to thank the authors of the papers for their submissions and for their contribution to the timely preparation of these proceedings, as well as for their high quality presentations and lively discussions during the workshop. At the same time, we would like our Program Committee for their hard work and for submitting their excellent reviews on time. For this edition of WEWST, out of 34 high quality submissions, we were able to accept 7 as full papers and 4 as short papers. We are also grateful to our keynote speaker, Schahram Dustdar, for sharing his insights on future research challenges of Web Services and putting them into perspective of the papers we have selected. His presentation was kindly supported by the IBM Zürich Research Lab and by the University of Lugano. We would also like to thank Monique Calisti and Whitestein Technologies AG, for the invaluable support in finding a suitable venue for publishing the workshop proceedings and Marc Herbstritt from Birkhäuser Verlag AG for fast tracking the WEWST proceedings through the publication process. Last but not least, we would like to thank the ECOWS conference organizers (Birgitta König-Ries, Claus Pahl, and Wolf Zimmermann) for their trust and availability to make this workshop a success.

Thomas Gschwind and Cesare Pautasso
Program Chairs WEWST'07

Rüschlikon – Lugano
May 2008

Organization

Program Chairs

Thomas Gschwind, IBM Research, Switzerland
Cesare Pautasso, University of Lugano, Switzerland

Program Committee

Luciano Baresi, Politecnico di Milano, Italy
Elisa Bertino, Purdue University, USA
Walter Binder, University of Lugano, Switzerland
David Breitgand, IBM Haifa, Israel
Christoph Bussler, BEA, USA
Fabio Casati, University of Trento, Italy
Malu Castellanos, HP, USA
Paco Curbera, IBM Watson, USA
Theo Dimitrakos, BT, UK
Jürgen Dunkel, FH Hannover, Germany
Schahram Dustdar, Vienna University of Technology, Austria
Daniela Grigori, Université de Versailles, France
Alexander Keller, IBM, New York, USA
Frank Leymann, University of Stuttgart, Germany
Mark Little, Red Hat, UK
Makoto Matsushita, Osaka University, Japan
Claus Pahl, Dublin City University, Ireland
Dumitru Roman, DERI Innsbruck, Austria
Ulf Schreier, University of Furtwangen, Germany

Reviewers

Federica Paci
Harumi Kuno

Introduction

This 2^{nd} volume on Emerging Web Services Technologies continues to follow the current research activities in the areas of Web Services and Service Oriented Architectures. By collecting the proceedings of the second Workshop of Emerging Web Services Technology 2007 it contains many examples of promising research activities cutting across a growing set of emerging technologies: service discovery and selection, service composition, extensions to BPEL, security, quality of service resource brokering and performance evaluation.

Part I opens the proceedings with an extended abstract outlining some of the research challenges ahead as presented in the very well received keynote given by Schahram Dustdar.

Part II continues this book with three chapters on service discovery and selection. The first chapter by Baldoni, Baroglio, Martelli, Patti, and Schifanella presents a novel approach to service discovery that goes beyond selection based on operation semantics matching by taking into account the context of the lookup as given by the choreography to which the service needs to conform. The second chapter by Stein, Barchewitz, and El Kharbili takes a look at the problem of service discovery from the perspective of business experts using the ARIS process modeling tool. The chapter gives a concrete example on how structural and semantics matching techniques are starting to make an impact in industrial applications and tools. The third chapter by Küster, Lausen, and König-Ries concludes this part by giving a thorough evaluation of current semantic service discovery efforts and outlining several important future research challenges.

Part III covers service composition from three different perspectives. The first chapter by Lécue, Silva, and Pires presents the SPICE framework for dynamic Web service composition. Services are chained taking into account their functional description in terms of input, output, preconditions and effects as well as non-functional properties. This results in a graph of semantic connections between the services. The second chapter by Lau and Tran contributes a novel component model tailored to recursive Web service composition. The third chapter by Charfi, Berbner, Mezini, and Steinmetz gives an in-depth discussion of requirements for the management of run-time aspects of Web service composition engines.

Part IV contains two extensions of the Business Process Execution Language (BPEL) standard. The first chapter by Habich, Richly, Preissler, Lehner, Grasselt, and Maier is about strenghtening BPEL to support data intensive applications. To supplement the existing 'by value' semantics, the authors introduce a new kind of data transition amenable to performance optimizations. The second chapter by Overdick foresees the application of BPEL to model the state of a RESTful Web services and proposes the necessary language extensions to deal with the identification and the uniform interface of resources.

Part V collects three different contributions related to Quality of Service (QoS) aspects. The first chapter by Gajek, Liao, Schwenk, and Möller addresses the problem of establishing a secure end-to-end communication channel for Web services. It leverages proven SSL/TLS mechanisms and applies them in the context of the exchange of SOAP messages. The second chapter by Zimeo and Ranaldo focuses on Grid resource brokering based on QoS guarantees. It presents a resource discovery framework that is capable of satisfing execution performance constraints of data-parallel applications virtualized as Web services. The final chapter by Pahl, Boskovic, and Hasselbring proposes to enhance model-driven engineering approaches with instrumentation constructs for model-based, empirical performance evaluation of service-oriented architectures.

Whitestein Series in Software Agent Technologies, 1–3

Emerging Web Services Technologies — Some Research Challenges Ahead

Schahram Dustdar

More than ever, computing devices are becoming more powerful and networked, organizational boundaries are dissolving, and underlying information systems become more complex, thus requiring higher degrees of autonomic behavior of the business processes and software services they support. In this keynote talk the main challenges towards building the required novel conceptual abstractions as well as needed technological implementations are presented and discussed.

In the past several decades the industrial landscape changed dramatically. Novel business models were increasingly introduced and successfully implemented. More recently, the vision of Service-oriented Architecture (SOA) aims at providing a model to allow realization of such novel, highly dynamic, adaptive, and composeable information systems and services for such business models and processes. SOAs are, in fact, mapping the real world unto the world of large-scale Internet-based information systems. Today we find many businesses and industries being "service-oriented". For example, telecommunications, financial services, healthcare, logistics, just to name a few. Those industries became "service-oriented" mainly through three factors: specialization, standardization, and scalability. All those factors can be also witnessed as being crucial in our educational systems. Standardization, in particular, is an important factor in the world of SOAs and business processes. In fact, it seems that as we see in the real world in many examples (e.g., Starbucks) we increasingly move to global standards of various products and services. In the Internet-world the same principle is applied to SOAs: Standards are being agreed upon and introduced (e.g., the Web services stack) and novel methods for building such global large-scale systems are being promoted:

The SOA for the top-down enterprise-scale approach to business process design and service composition (build once and use many times), and more recently, the service mashup approach (build once and use once), for the bottom-up end-user (consumer) driven approach to service composition. Service mashups have some additional characteristics, such as more or less concurrent design and execution, higher degree of user participation, and an overall agile approach to the development process.

Why are those approaches to service composition and business process design and management relevant at all? Why is it not enough to use workflow management systems? Or is it enough? Well, in this paper, I argue that those traditional approaches increasingly do not work. The reasoning is as follows: Throughout the last decades we have seen that organizational boundaries increasingly became fuzzy. Novel business alliances, including mergers and acquisitions, are occurring. Such partnerships happen more often and faster than previously. Furthermore, partnerships need to be highly dynamic and flexible, often depending on special cases and on-demand policies. In technical terms we can say that there is increasingly a need for information systems integration, however, the assumptions as we knew them from the area of workflow management systems (e.g., first you model, then you execute; after exceptions occur, remodel your process and enact again) do not hold any longer due to the requirements of highly dynamic, flexible and inter-connected organizations and people including the products and services they offer, provide and produce. The distinction between design (model or built) time and run time is starting to become obsolete. We need to spend more energy on analyzing finer "granularities" of those "times".

In this keynote I discussed the technical foundations, state-of-the art, and assumptions as well as implications with regard to current technology trends and summariz lessons-learned from four areas which are crucial to the topic of this talk, i.e., Infrastructure Evolution, Software Evolution, Process Evolution, and Teamwork Evolution.

About the Speaker

Schahram Dustdar is Full Professor of Computer Science with a focus on Internet Technologies heading the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology (TU Wien) where he is director of the Vita Lab. He is also Honorary Professor of Information Systems at the Department of Computing Science at the University of Groningen (RuG), The Netherlands. He is Chair of the IFIP Working Group 6.4 on Internet Applications Engineering and a founding member of the Scientific Academy of Service Technology.

He received his M.Sc. (1990) and PhD. degrees (1992) in Business Informatics (Wirtschaftsinformatik) from the University of Linz, Austria. In April 2003 he received his Habilitation degree (Venia Docendi in Angewandte Informatik) for his work on Process-aware Collaboration Systems - Architectures and Coordination Models for Virtual Teams. His work experience includes several years as the founding head of the Center for Informatics (ZID) at the University of Art and Industrial Design in Linz (1991-1999), Austrian project manager of the MICE EU-project (1993 - 97), and director of Coordination Technologies at the Design Transfer Center in Linz (1999 - 2000). While on sabbatical leave he was a post-doctoral research scholar (Erwin-Schrödinger scholarship) at the London School of Economics (Information Systems Department) (1993 and 1994), and a visiting

research scientist at NTT Multimedia Communications Labs in Palo Alto, USA during 1998.

From 1999–2007 he worked as the co-founder and chief scientist of Caramba Labs Software AG (CarambaLabs.com) in Vienna (acquired by Engineering NetWorld AG), a venture capital co-funded software company focused on software for collaborative processes in teams. Caramba Labs was nominated for several (international and national) awards: World Technology Award in the category of Software (2001); Top-Startup companies in Austria (Cap Gemini Ernst & Young) (2002); MERCUR Innovationspreis der Wirtschaftskammer (2002). Currently, Prof. Dustdar is on the management board of the Association of the alumni of the TU Wien.

He has published more than 160 scientific papers as conference-, journal-, and book contributions. He has written 3 academic books as well as one professional book. His latest book, co-authored with H. Gall and M. Hauswirth, is on software architectures for distributed systems (2003), Springer-Verlag. In 1997 he co-authored a book on Multimedia Information Systems, Kluwer and co-edited the book Telekooperation in Unternehmen, Gabler Verlag. He has published in various journals including ACM Transactions on Internet Technology, ACM Transactions on the Web, Distributed and Parallel Databases, Data and Knowledge Engineering, Journal of Grid Computing, WWW Journal, IEEE Multimedia, Business Process Management Journal, Journal of Systems Architecture, Journal of Organizational Computing, Kluwer Multimedia Tools and Applications, Wirtschaftsinformatik, and Journal of Computing and Information Technology. He co-organized several scientific workshops and conferences (e.g., ICSOC 2007, BPM 2006, DiSD 2005 colocated with RE; Teamware colocated with SAINT; CSSE colocated with ASE; UMICS 2003, 2004, 2005, 2006, colocated with CAiSE; DMC 2003, 2004, 2005, 2006 colocated with IEEE WETICE) and has been serving on more than 200 international program committees as well as on editorial boards of 10 scientific journals. His research interests include collaborative computing, workflow systems, Internet technologies, software architecture, distributed systems, distributed multimedia systems, and mobile collaboration systems. He is charter member of the Association of Information Systems (AIS), member of the IEEE Computer society, ACM, GI, and Austrian Computer Society. He was an invited expert evaluator for the IST 6th Framework (FP6) of the European Commission as well as an invited expert for the 7th Framework roadmap definitions for some working groups. He has been a scientific reviewer for a number of National Science Foundations (e.g., DFG (Germany), NWO (Netherlands), EPSRC (UK), SFI (Ireland), NSERC (Canada)).

Schahram Dustdar
Technische Universität Wien
Argentinierstrasse 8/184-1
A-1040 Wien, Austria
e-mail: dustdar@infosys.tuwien.ac.at

Whitestein Series in Software Agent Technologies, 5–22

Service selection by choreography-driven matching

Matteo Baldoni, Cristina Baroglio, Alberto Martelli,
Viviana Patti, and Claudio Schifanella

Abstract. The greater and greater quantity of services that are available over the web causes a growing attention to techniques that facilitate their reuse. A web service specification can be quite complex, including various operations and message exchange patterns. In this work, we focus on the problem of retrieving a web service, which can play a given choreography role, preserving at the same time a condition of interest (the goal for which the service is sought). We show that current semantic matchmaking techniques do not guarantee goal preservation. We also show an approach for overcoming these limits, which exploits the choreography definition. This work is based on an action-based representation of the operations of a service: each operation is described in terms of its preconditions and effects, without taking into account the ontology layer which is not functional to the aims of the work.

Mathematics Subject Classification (2000). Primary 68Q60; Secondary 68T27.

Keywords. Web services, choreographies, semantic matchmaking, reasoning about goals.

1. Introduction

Web services have a platform-independent nature, that endeavors enterprises to develop new business processes by combining existing services, retrieved over the web [21]. In the perspective of *service reuse*, the ability of retrieving services according to particular needs is crucial. Of course, it is unlikely to discover services that perfectly match a specification, some degree of flexibility is necessary. Nowadays, service retrieval is basically performed through registries like UDDI [26], where service advertisements are published. A common choice is to describe services by WSDL [31] specifications. In this context, retrieval cannot yet be accomplished automatically as well as desired because the representations used and the discovery mechanisms are semantically poor.

The need of adding a semantic layer to service descriptions brought to initiatives like the development of the language OWL-S [22] and the development of the Web Service Modeling Ontology (WSMO) [12]. In the semantic approach a richer annotation, aimed at representing the so called IOPEs (inputs, outputs, preconditions and effects of the service), is used. Inputs and outputs are usually described in terms taken from a public ontology, while preconditions and effects are often expressed by means of logic representations. The WSMO model is slightly different: here every service has also associated a logical formula, known as the *goal* of the service, which is matched with the request during the search. The goal captures the purposes of the service while the request represents the goal of the querier: the selection is performed only when the two match. Moreover, recently an extension of WSDL, called SAWSDL, that enables the use of semantic annotations has been proposed [27]. As a difference with the previous proposals, SAWSDL allows to semantically enrich only the definition of input and output parameters, by relating them with ontological concepts.

Semantic annotation allows the discovery of services, whose descriptions *do not exactly match* with the corresponding queries. Intuitively, they allow the retrieval of services, which have been developed for a (slightly) different purpose but that can however be used for the aims of the current query. Therefore, these techniques facilitate software reuse. So-called *semantic matchmaking techniques* (e.g. [23, 18, 12]) mainly exploit forms of ontological reasoning. Many of these proposals are inspired by the seminal work of Zaremski and Wing [32] for software components match, who propose a formal specification to describe and compare software components. They define various flavors of relaxed match, that capture the notions of *generalization*, *specialization*, and *substitutability*; the best-known of these relaxed matches is the *plugin match*. Specifications are given in terms of pre- and post-conditions, written as predicates in first-order logic. Already in this work, the goal was to identify software components that could be reused in a context that was not the one for which they were originally developed.

Semantic matchmaking focuses on the discovery of single services, in the sense that a service is considered as corresponding to a *single operation*. In general, however, the use of a web service implies the execution of a *sequence of operations* in a particular *order*, which might even involve other services [1]: for instance, the clients of a supplier web service have to identify themselves, request item prices and delivery time, and so on. In order for the interaction to be successful, the message exchange must obey some constraints: if they are not satisfied the service will be unable to process the messages and will return an error. To allow the interaction, web services exhibit *interfaces* (port-types) which gather various operations that are logically related. Moreover, it is possible to specify the order in which messages are to be exchanged by means of languages like WSCI [29] and, at a lower level of detail, WSDL message exchange patterns [30].

On the other hand, the need of describing *compositions* of services, which have to interact according to (complex) patterns of interaction, ruled by conversation protocols, has lead to the development of choreography languages like WS-CDL

[28]. WS-CDL is aimed at describing collaborations between any type of participant independently from the programming model used by its implementation. A WS-CDL specification can be seen as a sort of contract, that specifies the ordering conditions and constraints that rule the message exchange. The description is done from a global point of view, encompassing the expected behavior of all the participants. Each participant is supposed to use the global definition to build and test solutions that conform to it.

The task of selecting a web service, that should play a role in a choreography (rather than using the choreography as the design of a new set of services), implies verifying two things: the *conformance* of the service to the specification of a role of interest, and that the use of that service allows the achievement of the *goal*, that caused its search. *Conformance* guarantees the interoperability of the service with the players of the other roles [25, 13, 10] by guaranteeing that the message exchange will produce correct and accepted conversations. The *goal* that caused the search of a service is a condition that should hold after the whole interaction has taken place. It is not tied to the descriptions of some service operation but it is a *global condition* that should hold in the final state, obtained after the conclusion of the conversation/interaction. The achievement of the goal depends on the operation sequence because each operation can influence the executability and the outcomes of the subsequent ones. Therefore, the matchmaking process, that is applied to discover services, should not only focus on local properties of the single operations, e.g. IOPEs, but it should also consider the global schema of execution, which is given by the choreography.

In [2] we have faced the conformance issue, proposing a conformance test that is based on a variant of bisimulation. In this work, we focus on the second problem: the selection of existing services that can play given choreography roles, preserving a condition of interest. In particular, we show that performing a match operation by operation, by applying the definitions in [32], *does not* preserve the global goal. We also show how to *overcome these limits* by exploiting the choreography definition. Actually, it is possible to extract from the choreography some information that can be used to bias the matching process so that the global goal will be preserved. To this aim, we exploit an action-based representation of the operations of a service: each operation is described in terms of its preconditions and effects, as in [3], without taking into account the ontology layer which is not functional to the aims of the work. This representation supplies the mechanisms and the tools for reasoning on compositions of services, as described in choreographies; in particular, it supplies a representation of states and an execution model that can be reasoned about.

The article is organized as follows. Section 2 introduces a simple representation for services, that is based on a declarative language, to abstract away from the details of implementation. Section 3 reports the main results of the work: we introduce the notions of conservative and of uninfluential substitution, and we show that it is possible to exploit the choreography to select services in such a way that a goal of interest is preserved. Throughout these sections we will use a same

running example (introduced little by little), that is centered on the interaction ruled by the simple *purchase-flight* protocol in Figure 1. Such protocol captures the interaction between a flight ticket seller and one of its clients. In the various sections we will see the how the protocol specification is given, in the declarative language that we have adopted, as well as some implementations associated to specific services, discussing about them. Related works (Section 4) and conclusions end the paper.

2. Using a declarative language to represent services

In this section, we briefly summarize the notation that we use to represent services, introduced in [3], and we discuss the problem of verifying a global goal. The notation is based on a logical theory for reasoning about actions and change in a modal logic programming setting. In this perspective, the problem of reasoning amounts either to build or to traverse a sequence of transitions between *states*. A state is a set of *fluents*, i.e., properties whose truth value can change over time, due to the application of actions. In general, we cannot assume that the value of each fluent in a state is known: we want to have both the possibility of representing unknown fluents and the ability of reasoning about the execution of actions on incomplete states. To explicitly represent unknown fluents, we use an epistemic operator $\mathbf{B}$, to represent the beliefs an entity has about the world: $\mathbf{B}f$ means that the fluent f is known to be true, $\mathbf{B}\neg f$ means that the fluent f is known to be false. A fluent f is undefined when both $\neg\mathbf{B}f$ and $\neg\mathbf{B}\neg f$ hold ($\neg\mathbf{B}f \wedge \neg\mathbf{B}\neg f$). For expressing that a fluent f is undefined, we write $u(f)$. Thus each fluent in a state can have one of the three values: *true*, *false* or *unknown*.

2.1. Service representation

A *service description* is defined as a triple $\langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle$, where $\mathbf{O}$ is a set of operations, $\mathbf{G}$ is a set of actions that allow to receive messages, and $\mathbf{P}$ (*policy*) is a description of the interactive behavior of the service. The name "policy" derives from the literature concerning conversation protocols [15].

- The set $\mathbf{O}$ contains the descriptions of a set of service operations. An operation is an atomic action. As such, it is described in terms of its *executability preconditions* and *effects*, the former being a set of fluents (introduced by the keyword **possible if**) which must be contained in the service state in order for the operation to be applicable, the latter being a set of fluents (introduced by the keyword **causes**) which will be added to the service state after the operation execution. Formalized in these terms, operations, when executed, trigger a revision process on the actor's beliefs. Since we describe web services from a *subjective* point of view (i.e. taking the perspective of a specific service, by representing and reasoning on the service policies), we distinguish between the case when the service is either the initiator or the servant of an operation by further decorating the operation name with a notation inspired by [7].

With reference to a specific service, $operation^{\gg}(interlocutor, content)$ denotes the fact that the service is the initiator of the operation (as in the case of "solicit-response" interactions), while $operation^{\ll}(interlocutor, content)$ denotes the fact that the service is the servant of the operation (as in the case of "request-response" interactions). Formally, an operation is represented as:

$$operation^{\{\gg \mid \ll\}}(interlocutor, content) \ \textbf{possible if} \ \{P_1, \ldots, P_t\}$$
$$operation^{\{\gg \mid \ll\}}(interlocutor, content) \ \textbf{causes} \ \{E_1, \ldots, E_n\}$$

where E_i, $i \in [1, n]$, and P_j, $j \in [1, t]$, denote respectively the fluents, which are expected as effect of the execution of an operation and the precondition to its execution, while *content* denotes possible additional data that is required by the operation.

As an example, let's consider search_flight, an operation of a flight reservation service, which is offered by a *seller* and can be invoked by a *client* to search information about flights with given departure and arrival locations. From the point of view of the client, the search_flight is represented as:

$$\text{search\_flight}^{\gg}(seller, Date, Start, Dest)$$
$$\textbf{possible if} \ \{\mathbf{B}Start, \mathbf{B}Dest, \mathbf{B}Date\}$$
$$\text{search\_flight}^{\gg}(seller, Date, Start, Dest)$$
$$\textbf{causes} \ \{\mathbf{B}will\_get\_offer\}$$

This notation captures the preconditions and the effects of the operation: the precondition is that the departure location is known ($\mathbf{B}Start$), that the destination is known ($\mathbf{B}Dest$), and that the day of departure is also known ($\mathbf{B}Date$); the effect is that after the execution the invoker (the client) expects that an offer will be sent ($\mathbf{B}will\_get\_offer$). Instead, from the point of view of the seller search_flight is represented as:

$$\text{search\_flight}^{\ll}(client, Date, Start, Dest) \ \textbf{possible if} \ \{\}$$
$$\text{search\_flight}^{\ll}(client, Date, Start, Dest)$$
$$\textbf{causes} \ \{\mathbf{B}requested\_flight(client, Date, Start, Dest)\}$$

From the point of view of the seller, search_flight is an operation to be offered to its interlocutor (the client). Its execution causes to acquire knowledge about the client's flight request.

Last but not least, a service can also have internal operations, which can be included in its policy but are not visible from outside. Each operation is represented again as an atomic action, specified by its *preconditions* and its *effects*. Formally, it is defined as:

$$\text{operation}(content) \ \textbf{causes} \ \{E_1, \ldots, E_n\}$$
$$\text{operation}(content) \ \textbf{possible if} \ \{P_1, \ldots, P_t\}$$

where E_i, $i \in [1, n]$, and P_j, $j \in [1, t]$, denote respectively the fluents, which are expected as effect of the execution of an operation and the precondition to its execution, while *content* denotes possible additional data that is required by the operation. Notice that such operations can also be implemented as

invocations to other services. As an example, here is the description of the eval_offer internal operation of a possible client for the flight-purchase interaction:

$\mathsf{eval\_offer}(Flight)$ **possible if** $\{\mathbf{B}offer(Flight)\}$
$\mathsf{eval\_offer}(Flight)$ **causes** $\{\mathbf{B}eval\_rst(Flight, Y)\}$

eval_offer applies when an offer for a flight is available and produces an evaluation that can be used for taking internal decision. The variable Y ranges over the set $\{business, no\_business\}$ depending on the kind of ticket that is being considered.

- Besides operations, we explicitly represent actions that allow the reception of information. We call them *get-answer* actions (set **G**). The range of possible answers is supposed to be finite, in the sense that the interlocutor is supposed to use a message out of a finite and predefined set of alternatives: if a different message is sent the service is not able to handle it. We imagine the reception of a piece of information as a "one-way" operation that is invoked over the recipient. Actually, for technical reasons in our formalization, each possible alternative answer corresponds to a different operation of this kind. Formally, they are represented as:

  $\mathsf{receive\_act}(interlocutor, content)$ **receives** I

  where *interlocutor* is the partner in the interaction, *content* is used to store the received message, and I is a set of alternative action invocations each allowing the reception of one of the alternative messages. As an example get_answer allows the reception of either a $\mathsf{not\_available}^{\ll}$ answer or of an offer through the execution of one of the two actions $\mathsf{not\_available}^{\ll}$ and $\mathsf{offer}^{\ll}$. The decision of which of the two actions will be executed is up to the interlocutor that decides which message to send.

  $\mathsf{get\_answer}(Seller)$ **receives**
  $[\mathsf{not\_available}^{\ll}(Seller)$ **or** $\mathsf{offer}^{\ll}(Seller, Flight))]$

  In this example we do not use the *content*.
- **P** encodes the behavior for the service; it is a collection of clauses of the kind:

  p_0 **is** $p_1, \ldots, p_n$

  where p_0 is the name of the procedure and p_i, $i = 1, \ldots, n$, is either an atomic action (operation), a *get-answer* action, a test action (denoted by the symbol ?), or a procedure call. Procedures can be recursive and are executed in a goal-directed way, similarly to standard logic programs, and their definitions can be non-deterministic as in Prolog. As an instance, here we report the booking procedure:

  $\mathsf{booking}(Seller, Date, Start, Dest)$ **is**
  $\mathsf{search\_flight}^{\gg}(Seller, Date, Start, Dest)$, $\mathsf{get\_answer}(Seller)$,
  $\mathbf{B}offer(not\_avail)$?
  $\mathsf{booking}(Seller, Date, Start, Dest)$ **is**
  $\mathsf{search\_flight}^{\gg}(Seller, Date, Start, Dest)$, $\mathsf{get\_answer}(Seller)$,

$\mathbf{B}offer(Flight)?,\ \mathsf{eval\_offer}(Flight),\ \mathsf{finalize}(Seller, Flight)$

It is defined by a set of two clauses, the former capturing the case when the ticket is not available, the latter the normal situation when an offer for a ticket is actually returned. In this case, the offer is evaluated and the purchase is finalized by invoking another procedure.

A *choreography* is made of a set of interacting *roles*, a role being a subjective view of the interaction that is encoded. When a service plays a role in a choreography, its policy will contain some operations which are not of the service itself but belong to some other role of the choreography, with which it interacts. In other words, $\mathbf{O}$ can be partitioned in two sets: a set of bound operations and a set of unbound operations, that must be supplied by some counterpart(s). Until the counterpart(s) service is (are) not defined, such operations will be those specified in the choreography. We assume that they are represented in a way that is homogeneous with the representation of operations, i.e. by means of preconditions and effects. The binding will be possible only when the partner in the interaction will be found. The fact that the former service is taking a given role in the choreography is due, in our proposal, to the fact that it knows that a certain goal condition will be true after the execution of the role. When a possible partner is identified for the latter role, after the binding has taken place, it is necessary to check if the goal condition is preserved. The reasons for which this could not happen are explained in the following section; hereafter, we formalize the notion of *substitution* that we interpret as the binding.

Let $S_d = \langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle$ a service description, and let $\mathbf{O}_u$ be a subset of $\mathbf{O}$, containing unbound operations that are to be supplied by a same counterpart S_i. Let $\mathbf{O}_{S_i}$ be the set of operations in S_i that we want S_d to use, binding them to $\mathbf{O}_u$. We represent the binding by the substitution $\theta = [\mathbf{O}_{S_i}/\mathbf{O}_u]$ applied to S_d, i.e.: $S_d\theta = \langle \mathbf{O}\theta, \mathbf{G}\theta, \mathbf{P}\theta \rangle$, where every element of $\mathbf{O}_u$ is substituted by/bound to an element of $\mathbf{O}_{S_i}$. Notice that not all elements of $\mathbf{O}_{S_i}$ are, instead, necessarily bound. An example is reported in *Example* 3.

Example 1. Let us introduce, as an example, a simple choreography (see Figure 1) that rules a flight reservation protocol, inspired to [24], with two roles: a *Buyer* and a *Seller*. The buyer sends a request to search for flights, specifying the departure location, the destination, and date. Depending on the seat availability, the seller can either refuse the request, or send the information regarding a specific flight. The buyer checks the offer, then, it either refuses (*n_ack*) or accepts it (*ack*). All the names on the arrows, e.g. *searchFlight* and *offer*, are specifications of the operations that the players must provide and perform.

Let us consider a service $b1$ that is conformant to the role *Buyer*. Following the proposed notation, we describe it as $\langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle$, where $\mathbf{P} = \{\mathsf{booking}, \mathsf{finalize}\}$, $\mathbf{O} = \{\mathsf{search\_flight}^{\gg}{}_u, \mathsf{not\_available}^{\ll}, \mathsf{eval\_offer}, \mathsf{offer}^{\ll}, \mathsf{ack}^{\gg}{}_u, \mathsf{n\_ack}^{\gg}{}_u\}$, $\mathbf{G} = \{\mathsf{get\_answer}\}$. Intuitively, we assume that $b1$ already checked to be able to play the *buyer* role, by proving that it owns both the internal (eval_offer) required

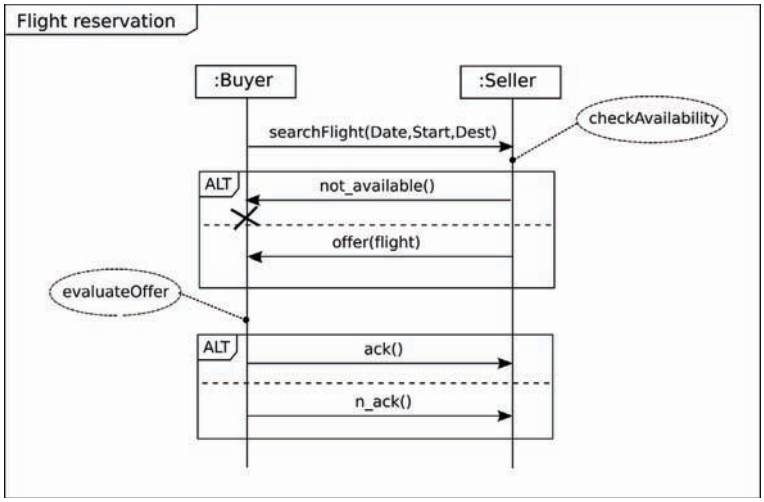


FIGURE 1. An example of a simple interaction protocol, for reserving a flight, expressed as a UML sequence diagram.

operations, and the ones foreseen by the protocol (not_available$^{\ll}$,offer$^{\ll}$). The checking can be performed by generalizing the approach in [4]. The procedures in **P** are described by the following clauses:

booking($Seller, Date, Start, Dest$) **is**
search_flight$^{\gg}{}_u$($Seller, Date, Start, Dest$), get_answer($Seller$),
B*offer*(not_avail)?
booking($Seller, Date, Start, Dest$) **is**
search_flight$^{\gg}{}_u$($Seller, Date, Start, Dest$), get_answer($Seller$),
B*offer*($Flight$)?, eval_offer($Flight$), finalize($Seller, Flight$)
finalize($Seller, Flight$) **is**
B*eval_rst*($Flight, business$)?, ack$^{\gg}{}_u$($Seller, Flight$)
finalize($Seller, Flight$) **is**
B*eval_rst*($Flight, no\_business$)?, n_ack$^{\gg}{}_u$($Seller, Flight$)

The only get_message action in **G** is described by:

get_answer($Seller$) **receives** [not_available$^{\ll}$($Seller$) **or** offer$^{\ll}$($Seller, Flight$)]

Finally, the operations in **O** are described as:

eval_offer($Flight$) **causes** {**B***eval_rst*($Flight, Y$)}
eval_offer($Flight$) **possible if** {**B***offer*($Flight$)}

search_flight$^{\gg}{}_u$($Seller, Date, Start, Dest$) **causes** {**B***will_get_offer*}
search_flight$^{\gg}{}_u$($Seller, Date, Start, Dest$) **possible if** {**B***start*, **B***dest*, **B***date*}

not_available$^{\ll}$($Seller$) **causes** {**B***offer*($not\_available$)}

$\mathsf{not\_available}^{\ll}(Seller)$ **possible if** $\{\}$

$\mathsf{offer}^{\ll}(Seller, Flight)$ **causes** $\{\mathbf{B}offer(Flight)\}$
$\mathsf{offer}^{\ll}(Seller, Flight)$ **possible if** $\{\}$

$\mathsf{ack}^{\gg}{}_u(Seller, Flight)$ **causes** $\{\mathbf{B}booked(Flight)\}$
$\mathsf{ack}^{\gg}{}_u(Seller, Flight)$ **possible if** $\{\}$

$\mathsf{n\_ack}^{\gg}{}_u(Seller, Flight)$ **causes** $\{\mathbf{B}\neg booked(Flight)\}$
$\mathsf{n\_ack}^{\gg}{}_u(Seller, Flight)$ **possible if** $\{\}$

where Y ranges on the set $\{business, no\_business\}$.

2.2. Reasoning on goals

In the outlined framework, it is possible to reason about goals by means of queries of the form:

$$Fs \ \textbf{after} \ p$$

where *Fs* is the goal (represented as a conjunction of fluents), that we wish to hold after the execution of a policy p. Checking if a formula of this kind holds corresponds to answering the query: "Is it possible to execute p in such a way that the condition *Fs* is true in the final state?". When the answer is positive, the reasoning process returns a sequence of atomic actions that allows the achievement of the desired condition. This sequence corresponds to an execution trace of the procedure and can be seen as a plan to bring about the *goal Fs*. Such a plan can be conditional because whenever a *get-answer* action is involved, none of the possible answers from the interlocutor can be excluded. In other words, the trace will contain a different execution branch for every option.

This form of reasoning is known as *temporal projection*. Temporal projection fits our needs because, as mentioned in the introduction, in order to perform the selection we need a mechanism that verifies if a goal condition holds after the interaction with the service has taken place. *Fs* is the set of facts that we would like to hold "after" p.

Let $S_d = \langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle$ be a service description. The application of temporal projection to $\mathbf{P}$ returns, if any, an execution trace, that makes a goal of interest become true. Let us, then, consider a procedure p belonging to $\mathbf{P}$, and denote by G the query *Fs* **after** p. Given a state S_0, containing all the fluents that we know as being true in the beginning, we denote the fact that G is successful in S_d by:

$$(\langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle, S_0) \vdash G$$

The execution of the above query returns as a side-effect an *execution trace* σ of p. The execution trace σ can either be *linear*, i.e. a terminating sequence $a_1, \ldots, a_n$ of atomic actions, or it can contain branches, that are due, as we have mentioned, to the presence of get-message actions.

Example 2 (Flight-purchase, second part). Let us suppose that the initial state of the service $b1$ is $S_0 = \{\mathbf{B}date, \mathbf{B}start, \mathbf{B}dest, \mathbf{B}smoking\_flight\}$, (all the other

fluents truth value is "unknown"). This means that $b1$ assumes a date, a departure location, an arrival location and that on the flight it is allowed to smoke. The goal of $b1$ is that the following condition holds:

$$G = \{\mathbf{B}booked(flight), \mathbf{B}smoking\_flight\} \textbf{ after } \mathsf{booking}(seller, date, start, dest)$$

Intuitively, the buyer expects that, after the interaction, it will have a reservation on a smoking flight.

By reasoning on its policy and by using the definitions of the unbound operations that are given by the choreography, $b1$ can identify an execution trace, that leads to a state where G holds:

$$\sigma = \mathsf{search\_flight}^{\gg}{}_u(seller, date, start, dest); \mathsf{offer}^{\ll}(seller, flight); \\ \mathsf{eval\_offer}(flight); \mathsf{ack}^{\gg}{}_u(seller, flight)$$

This is possible because in a declarative representation specifications are executable. Moreover notice that this execution does not influence the belief about the smoking flight, which persists from the initial through the final state and is not contradicted.

3. Goal-preserving match

When the matching process is applied for selecting a service that should play a role in a (partially instantiated) choreography, the desire is that the substitution (of the service operations to the specifications contained in the choreography) preserves the properties of interest. Let us formalize this notion.

Definition 3.1 (Conservative substitution). Let us consider a service $S_i = \langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle$ which plays a role R_i in a given choreography, and a query G such that, given an initial state S_0,

$$(\langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle, S_0) \vdash G \text{ w.a. } \sigma$$

Consider a substitution $\theta = [\mathbf{O}_{S_j}/\mathbf{O}^{\sigma}_{u(R_j)}]$, where $\mathbf{O}^{\sigma}_{u(R_j)} = \{o_u \in \mathbf{O} \mid o \text{ occurs in } \sigma\}$ is the set of all unbound operations that refer to another role R_j, $j \neq i$, of the same choreography, that are used in the execution trace σ. θ is conservative when the following holds:

$$(\langle \mathbf{O}\theta, \mathbf{G}\theta, \mathbf{P}\theta \rangle, S_0) \vdash G \text{ w.a. } \sigma\theta$$

In the above definition, θ can be any kind of association between the operations of a service with the operations described in a choreography. In practice is the result of a matching process. In the literature it is possible to find many match algorithms, many of them (in the case of semantic web services) are grounded into the work by Zaremski and Wing [32], mentioned in the introduction.

Zaremski and Wing propose a formal specification to describe the behavior of software components, and to determine if two components match. Each software

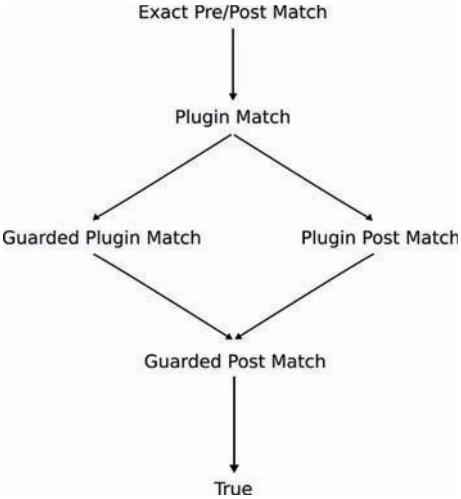


FIGURE 2. The lattice of the different local matches: on top the strongest.

component has precondition $\mathsf{Precs}(s)$ and postcondition $\mathsf{Effs}(s)$. Their specifications are matched against a requirement, coherently specified as having precondition $\mathsf{Precs}(r)$ and postcondition $\mathsf{Effs}(r)$. Five kinds of relaxed match between r and s are defined, that we rephrase hereafter, to adapt them to our framework:

- EM (*Exact Pre/Post Match*): $\mathsf{Precs}(r) = \mathsf{Precs}(s) \wedge \mathsf{Effs}(r) = \mathsf{Effs}(s)$
- PIM (*Plugin Match*): $\mathsf{Precs}(r) \supseteq \mathsf{Precs}(s) \wedge \mathsf{Effs}(s) \supseteq \mathsf{Effs}(r)$
- POM (*Plugin Post Match*): $\mathsf{Effs}(s) \supseteq \mathsf{Effs}(r)$
- GPIM (*Guarded Plugin Match*): $\mathsf{Precs}(r) \supseteq \mathsf{Precs}(s) \wedge ((\mathsf{Precs}(s) \cup \mathsf{Effs}(s)) \supseteq \mathsf{Effs}(r))$
- GPOM (*Guarded Post Match*): $((\mathsf{Precs}(s) \cup \mathsf{Effs}(s)) \supseteq \mathsf{Effs}(r))$

Exact pre/post match states the equivalence of r and s. *Plugin match* is weaker: s must only be behaviorally equivalent to r when plugged-in to replace r. *Plugin post match* relaxes the former: only the postcondition is considered. *Guarded matches* focus on guaranteeing that the desired postcondition holds when the precondition of s holds, not necessarily in general. The different matches can be organized according to a lattice [32], that we have reported in Fig. 2. For short, we will respectively denote by θ_{EM}, θ_{PIM}, θ_{POM}, θ_{GPIM}, θ_{GPOM}, the substitutions obtained by applying the five degrees of match.

It is immediate to see that any substitution, obtained by applying the exact pre/post match, satisfies Definition 3.1. However, this is not true for the other kinds of match. Let us show this with the help of a simple example.

Example 3. The buyer service $b1$ (see previous examples) is looking for another service, which can play the role of the *Seller*, to reserve a flight seat. This service must provide a set of operations that will substitute the unbound operations of

the buyer role. Let us choose the *plugin match* as matching rule. Let us consider the candidate $s1$, a service that is conformant to the protocol w.r.t. the role *Seller*. The set of operations of the seller represented in the knowledge base of the buyer includes the following definition for operation $\mathsf{search\_flight}^{\gg}$:

$\mathsf{search\_flight}^{\gg}(Seller, Date, Start, Dest)$
possible if $\{\mathbf{B}start,\ \mathbf{B}dest,\ \mathbf{B}date\}$
$\mathsf{search\_flight}^{\gg}(Seller, Date, Start, Dest)$
causes $\{\mathbf{B}will\_get\_offer,\ \mathbf{B}\neg smoking\_flight\}$

while all the other operations are defined exactly as in Example 1.

By applying the *plugin match*, we obtain the substitution θ_{PIM}, which includes, among the others, also $[\mathsf{search\_flight}^{\gg}/{\mathsf{search\_flight}^{\gg}}_{u}]$ [1]. By applying the substitution θ_{PIM} we obtain the set of policies $\mathbf{P}\theta_{PIM}$:

$\mathsf{booking}(Seller, Date, Start, Dest)$ **is**
$\mathsf{search\_flight}^{\gg}(Seller, Date, Start, Dest)$, $\mathsf{get\_answer}(Seller)$,
$\mathbf{B}offer(not\_avail)?$
$\mathsf{booking}(Seller, Date, Start, Dest)$ **is**
$\mathsf{search\_flight}^{\gg}(Seller, Date, Start, Dest), \mathsf{get\_answer}(Seller)$,
$\mathbf{B}offer(Flight)?; \mathsf{eval\_offer}(Flight); \mathsf{finalize}(Seller, Flight)$

$\mathsf{finalize}(Seller, Flight)$ **is**
$\mathbf{B}eval\_rst(Flight, business)?; \mathsf{ack}^{\gg}(Seller, Flight)$
$\mathsf{finalize}(Seller, Flight)$ **is**
$\mathbf{B}eval\_rst(Flight, no\_business)?; \mathsf{n\_ack}^{\gg}(Seller, Flight)$

By using this policy, the query $(\langle \mathbf{O}\theta_{PIM}, \mathbf{G}\theta_{PIM}, \mathbf{C}, \mathbf{P}\theta_{PIM}\rangle, S_0) \vdash G$ fails: in fact, the additional effect $\mathbf{B}\neg smoking\_flight$ of the service $\mathsf{search\_flight}^{\gg}$ prevents the buyer to achieve a part of its goal, i.e. to book a *smoking* flight.

Theorem 3.2. *The class of PIM, POM, GPIM and GPOM substitutions are not conservative.*

Proof. The proof is given by the counterexample in Example 3. In fact, θ, besides being a PIM substitution, is also an instance of all the other kinds of substitution that we have listed, i.e. it is also a POM, a GPIM, and a GPOM substitution. □

In order for a substitution to be conservative, it must take into account also the *overall structure*, encoded by the choreography. The locality of the matches used in the matchmaking phase, indeed, seriously limits the possibility of re-using services by selecting and composing them in an automatic way.

In the remainder of this section, we focus on the *plug-in match*. The plugin match is one of the most used matches and it immediately follows the exact match in the lattice (it is the strongest of the flexible matches). We show how to enrich it so to allow the construction of conservative substitutions. To this aim, we take into

[1] For the sake of brevity, we omit to specify the substitutions when the operations exactly match the specifications.

account the *dependencies* between actions, which produce as effects fluents, that are used as preconditions by subsequent action. Intuitively, the idea is to verify that the "causal chain" which allows the execution of the sequence of actions, is not broken by the differences between capabilities/services and requirements, as instead happens in the example. The obvious hypothesis is that we have a choreography and that we know that it allows to achieve the goal of interest, i.e. that there is an execution σ of the role specification, which allows the achievement of the goal. We will use this trace for defining the additional properties for the match.

Let us start by introducing the notions that define dependencies between actions and dependency sets for fluents. Consider a service description $S = \langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle$, and suppose that, given the initial state S_0, the goal $G = Fs$ **after** p succeeds, thus obtaining as answer the successful sequence of actions $\sigma = a_1; a_2; \ldots; a_n$, which is an execution trace of p.[2] We denote by $\overline{\sigma}$ the sequence of actions $a_0; a_1; a_2; \ldots; a_n; a_{n+1}$, where a_0 and a_{n+1} are two *fictitious* actions that will be used respectively to represent the initial state S_0 and the set of fluents Fs, which must hold after σ. That is, we assume a_0 has no precondition and $\mathsf{Effs}(a_0) = S_0$, and that a_{n+1} has no effect but $\mathsf{Precs}(a_{n+1}) = Fs$.

Consider two indexes i and j, such that $j < i$, $i, j = 0, \ldots, n+1$. We say that *in* $\overline{\sigma}$ the action a_i *depends on* a_j *for the fluent* $\mathbf{B}l$, written $a_j \rightsquigarrow_{\langle \mathbf{B}l, \overline{\sigma} \rangle} a_i$, iff $\mathbf{B}l \in \mathsf{Effs}(a_j)$, $\mathbf{B}l \in \mathsf{Precs}(a_i)$, and there is not a k, $j < k < i$, such that $\mathbf{B}l \in \mathsf{Effs}(a_k)$. Given a fluent $\mathbf{B}l$ and a sequence of actions σ, we can, therefore, define the *dependency set* of $\mathbf{B}l$ as $\mathsf{Deps}(\mathbf{B}l, \sigma) = \{(j, i) \mid a_j \rightsquigarrow_{\langle \mathbf{B}l, \overline{\sigma} \rangle} a_i\}$.

Let $[s/o_u]$ be a specific substitution of a service operation s to an unbound operation o_u, that is contained in θ_{PIM}, we say that a fluent $\mathbf{B}l \in \mathsf{Effs}(s) - \mathsf{Effs}(o_u)$ (i.e. an additional effect of s w.r.t. the effects of o_u) is an *uninfluential fluent* w.r.t. the sequence $\sigma\theta_{PIM}$ iff for all pairs $(j, i) \in \mathsf{Deps}(\mathbf{B}\neg l, \sigma)$, identifying by k the position of o_u in σ, we have that $k < j$ or $i \leq k$, Intuitively, this means that the fluent will not break any dependency between the actions which involve the inverse fluent because either it will be overwritten or it will appear after its inverse has already been used. Note that σ and $\sigma\theta_{PIM}$ have the same length and are identical as sequences of actions but for the fact that in the latter the selected service operations substitute unbound operations. For this reason, we can reduce to reasoning on σ for what concerns the action positions.

A substitution θ_{PIM} is called *uninfluential* iff for any substitution $[s/o_u]$ in θ_{PIM}, all beliefs in $\mathsf{Effs}(s) - \mathsf{Effs}(o_u)$ are uninfluential fluents w.r.t. σ. Now we are in position to prove that a substitution which exploits the *plugin match* and which is also *uninfluential*, is conservative.

Theorem 3.3. *Let us consider a service* $S_i = \langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle$ *which plays a role* R_i *in a given choreography, and a query* G *such that, given an initial state* S_0*,*

$$(\langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle, S_0) \vdash G \text{ } w.a. \text{ } \sigma$$

[2] In the following we focus on linear plans. Conditional plans can be tackled by considering each path separately.

Consider an uninfluential substitution $\theta_{PIM} = [\mathbf{O}_{S_j}/\mathbf{O}^{\sigma}_{u(R_j)}]$, *where* $\mathbf{O}^{\sigma}_{u(R_j)} = \{o_u \in \mathbf{O} \mid o \text{ occurs in } \sigma\}$ *is the set of all unbound operations that refer to another role* R_j, $j \neq i$, *of the same choreography, that are used in the execution trace* σ. *Then, the following holds:*

$$(\langle \mathbf{O}\theta_{PIM}, \mathbf{G}\theta_{PIM}, \mathbf{P}\theta_{PIM} \rangle, S_0) \vdash G \text{ w.a. } \sigma\theta_{PIM}$$

Proof. The proof is by absurd and it uses the proof theory introduced in [6]. Let us assume that $(\langle \mathbf{O}, \mathbf{G}, \mathbf{P} \rangle, S_0) \vdash G$ w.a. σ but $(\langle \mathbf{O}\theta_{PIM}, \mathbf{G}\theta_{PIM}, \mathbf{P}\theta_{PIM} \rangle, S_0) \nvdash G$ w.a. $\sigma\theta_{PIM}$. Since, by hypothesis, for any substitution $[o/o_u]$ in θ_{PIM}, $\mathsf{Effs}(o) \subseteq \mathsf{Effs}(o_u)$ holds, there exists a fluent F such that $a_0, a_1, \ldots, a_{i-1} \vdash F$ but $(a_0, a_1, \ldots, a_{i-1})\theta_{PIM} \nvdash F$, where $\sigma = a_0, a_1, \ldots, a_{i-1}, a_i, \ldots, a_n$ and $F \in \mathsf{Precs}(a_i)$. Now, since $a_0, a_1, \ldots, a_{i-1} \vdash F$, there exists $j \leq i-1$, such that $a_0, a_1, \ldots, a_j \vdash F$ and $F \in \mathsf{Effs}(a_j)$ but $(a_0, a_1, \ldots, a_j)\theta_{PIM} \nvdash F$, that is $F \notin \mathsf{Effs}(a_j\theta_{PIM})$. This is absurd due to the hypothesis that θ_{PIM} is an uninfluential substitution. □

Example 4. Let us now consider the goal and the service description specified in Example 1, and let us also consider an interlocutor $s2$, that is conformant to the role *Seller*. For what concerns the operation $\mathsf{ack}^{\gg}{}_u$ and $\mathsf{n\_ack}^{\gg}{}_u$, the service $s2$ offers descriptions that exactly match the specification in Example 1. Instead for what concerns $\mathsf{search\_flight}^{\gg}{}_u$, it offers the following description:

$\mathsf{search\_flight}^{\gg}(Seller, Date, Start, Dest)$
  possible if $\{\mathbf{B}start,\ \mathbf{B}dest,\ \mathbf{B}date\}$
$\mathsf{search\_flight}^{\gg}(Seller, Date, Start, Dest)$
  causes $\{\mathbf{B}will\_get\_offer,\ \mathbf{B}veg\_meals\}$

Differently than in case of $s1$, this service does not compromise the achievement of the goal, even though it provides some additional information ($\mathbf{B}veg\_meals$). This information is not used in the interaction that we are considering but we must take into account the fact that $s2$ might be conformant also to other protocols, in which this information is relevant. It is realistic that the service will not be re-implemented each time if not strictly necessary.

The verification that a substitution is uninfluential involves the derivation σ, and it is based on checking whether the chains of dependencies between actions for the various fluents are not interrupted by some opposite fluent. Obviously, if the domain is such that no fluent, once asserted, can be negated, any θ_{PIM} will be conservative. This can be verified statically on the choreography and the set of unbound operations, by checking that every fluent (that appears as effect of some action) is always positive or negative, including the initial state and the goal in the verification. Indeed, the application domains in which actions produce *knowledge* are of this kind.

4. Conclusion and related works

In this work we have studied the relation between the matchmaking rules and the achievement of a goal in a choreography, within the process of selecting a service for playing a role. We have shown that, when the adoption of a role is due to the desire of reaching a goal, the matches performed on single operations (but the exact match) are not adequate and it is necessary to introduce a verification that takes into account the context given by the choreography. Afterwards, we have presented an extension of the plugin match that takes into account also the choreography. To the best of our knowledge, the selection of a service based on a kind of match that takes into account *context of application* of the sought services (i.e. the choreography in which it will be immersed) has not been yet tackled in the literature, with the only exception of the work by Biswas [8]. Biswas proposes to enrich service descriptions with constraints, i.e. conditions that hold during the execution of the service. Given a specification of a desired composed service, in BPEL or in OWL-S, a discovery process is enacted to identify the services to assemble. The constraints associated to them are used to build the overall constraints of the composition, which is then checked against the constraints given by the user, to see if the composition satisfies the user's needs. This is a bottom-up approach, aimed at verifying some properties of the composition which are not captured by the IOPE analysis.

The literature related to matchmaking is wide and it is really difficult to be exhaustive. The matches proposed in [32] have inspired most of the semantic matches for web service discovery. Amongst them, Paolucci et al. [23] propose four degrees of match (exact, plugin, subsumes, and fail) that are computed on the ontological relations of the outputs of an advertisement for a service and a query. This approach tackles DAML-S representations, in which services are described by means of inputs and outputs. This approach is refined in [18], a work that describes a service matchmaking prototype, which uses a DAML-S based ontology and a Description Logic reasoner to compare ontology-based service descriptions, given in terms of input and output parameters. The matchmaking process, like in [23], produces a discrete scale of degrees of match (Exact, PlugIn, Subsume, Intersection, Disjoint).

WSMO (Web Service Modeling Ontology) [12] is an organizational framework for semantic web services. As such, it does not suggest a specific matching rule, which is up to the specific implementations. However, the authors propose in [17] an approach that is based on [32] and on [18]. More recently, a WSMO matchmaker has been proposed in [16], which combines several aspects: type matching, relation matching, constraint matching, parameter matching, intentional matching. Last but not least, in [20] a multi-level evaluation model is proposed, for deciding whether two services are composable. This is done through four levels of control (quality, dynamic semantics, static semantics, and syntax). Dynamic semantics is the name given to the matches of [32].

The idea of synthesizing a policy from an abstract specification is also stated in [11], where it is observed that services are often conceived so as to be delivered individually, while there is a growing need of reusing this software, either by composing services or by tailoring a composition to some specific client. This direction has been suggested in [21], where a UML specification of a business process was used to abstract the description of a composition away from the specification of the composed services. This abstract specification defines a *model*, used for driving the retrieval and the composition task.

Works like [24, 9] propose approaches for goal-driven service composition based on planning. However, the task is accomplished without reference to any choreography. In particular, in [24] the composition and the semantic reasoning phases (carried on on inputs and outputs) are separated and the latter is performed on a local basis only. In [14, 19] web services are composed by composing their interaction protocols in a social framework, by means of a temporal logic.

The next step of this research will be to test the presented method, by implementing it in a real system and applying it to real cases. In particular, we are exploring the possibility to design a choreography-driven matchmaking framework by relying on existing description languages, such as OWL-S, for what concerns the semantic service representation, and WS-CDL+C [5] for what concerns the choreography specification. Moreover, so far we have not yet tackled the integration of ontological reasoning in our work. This is surely an interesting extension that we will face soon; actually, many proposals for semantic matchmaking base upon the same relaxed match that we have used, and we expect similar results.

Acknowledgment

This research has partially been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6^{th} FP project REWERSE number 506779 (cf. http://rewerse.net), and by MIUR PRIN 2005 "Specification and verification of agent interaction protocols" national project. Claudio Schifanella is partially supported by the fellowship program "Fondazione CRT - Progetto Lagrange" (cf. http://www.progettolagrange.it).

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services Concepts, Architectures and Applications*. Springer-Verlag, Berlin, 2004.

[2] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. A priori conformance verification for guaranteeing interoperability in open environments. In *Proc. of the 5th International Conference on Service Oriented Computing (ICSOC 2006)*, volume 4294 of *LNCS*, pages 339–351. Springer, 2006.

[3] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about interaction protocols for customizing web service selection and composition. *J. of Logic and Algebraic Programming*, 70(1):53–73, 2007.

[4] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Goal preservation by choreography-driven matchmaking. In *Proc. of the Third International Workshop on Engineering Service-Oriented Applications: Analysis, Design and Composition, WESOA 2007, in conjunction with ICSOC 2007*, pages 77–88, Vienna, Austria, September 2007.

[5] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Reasoning on choreographies and capability requirements. *International Journal of Business Process Integration and Management, IJBPIM*, 2007.

[6] M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Programming Rational Agents in a Modal Action Logic. *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*, 41(2-4):207–257, 2004.

[7] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Synthesis of Underspecified Composite e-Service bases on Atomated Reasoning. In *Proc. of ICSOC04, 2nd International Conference on Service Oriented Computing*, pages 105–114, 2004.

[8] D. Biswas. Web services discovery and constraints composition. In *Proc. of the 1st Int. Conf. on Web Reasoning and Rule Systems, RR 2007*, volume 4524 of *LNCS*, pages 73–87. Springer, 2007.

[9] J. Bryson, D. Martin, S. McIlraith, and L. A. Stein. Agent-based composite services in DAML-S: The behavior-oriented design of an intelligent semantic web. In *Agent-Based Composite Services in DAML-S: The Behavior-Oriented Design of an Intelligent Semantic Web, Web Intelligence*. Springer-Verlag, 2002.

[10] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration: A synergic approach for system design. In *Proc. of 4th International Conference on Service Oriented Computing (ICSOC 2005)*, volume 3826 of *LNCS*, pages 228–240. Springer, 2005.

[11] F. Casati and M. C. Chien. Dynamic and adaptive composition of e-services. *Information Systems*, 26:143–163, 2001.

[12] D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, D. Roman, and A. Polleres. *Enabling Semantic Web Services : The Web Service Modeling Ontology*. Springer, 2007.

[13] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based analysis of obligations in web service choreography. In *Proc. of IEEE International Conference on Internet&Web Applications and Services 2006*, 2006.

[14] L. Giordano and A. Martelli. Web Service Composition in a Temporal Action Logic. In *Proc. of 4th Int. Work. on AI for Service Composition*, 2006.

[15] M. P. Huget and J.L. Koning. Interaction Protocol Engineering. In H.P. Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *LNAI*, pages 179–193. Springer, 2003.

[16] F. Kaufer and M. Klusch. WSMO-MX: A logic programming based hybrid service matchmaker. In *ECOWS '06: Proc. of the European Conference on Web Services*, pages 161–170, Washington, DC, USA, 2006. IEEE Computer Society.

[17] U. Keller, R. Laraand A. Polleres, I. Toma, M. Kifer, and D. Fensel. D5.1 v0.1 WSMO web service discovery. Technical report, WSML deliverable, 2004. Available at http://www.wsmo.org/TR/d5/d5.1/v0.1/.

[18] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 331–339, New York, NY, USA, 2003. ACM.

[19] A. Martelli and L. Giordano. Reasoning About Web Services in a Temporal Action Logic. In *Reasoning, Action and Interaction in AI Theories and System*, number 4155 in LNAI, pages 229–246. Springer, 2006.

[20] B. Medjahed and A. Bouguettaya. A multilevel composability model for semantic web services. *IEEE Trans. on Knowledge and Data Engineering*, 17(7):954–968, 2005.

[21] B. Örrens, J. Yang, and M.P. Papazoglou. Model driven service composition. In *ICSOC 2003, Proceedings of the First International Conference on Service-Oriented Computing, 2003*, volume 2910 of *LNCS*, pages 75–90. Springer, 2003.

[22] OWL-S Coalition. http://www.daml.org/services/owl-s/.

[23] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic matching of web services capabilities. In *Proc. of ISWC'02*, pages 333–347. Springer, 2002.

[24] M. Pistore, L. Spalazzi, and P. Traverso. A minimalist approach to semantic annotations for web processes compositions. In *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, June, 2006, Proceedings*, volume 4011 of *LNCS*, pages 620–634. Springer, 2006.

[25] S. K. Rajamani and J. Rehof. Conformance checking for models of asynchronous message passing software. In *Proc. of 14th International Conference on Computer Aided Verification, CAV 2002*, volume 2404 of *LNCS*, pages 166–179. Springer, 2002.

[26] UDDI, Universal Description, Discovery and Integration. http://www.uddi.org/.

[27] W3C. Semantic Annotations for WSDL Working Group. http://www.w3.org/2002/ws/sawsdl/.

[28] WS-CDL. http://www.w3.org/tr/ws-cdl-10/.

[29] WSCI, Web Service Choreography Interface. http://www.w3.org/tr/wsci.

[30] WSDL Message Exchange Patterns. http://www.w3.org/tr/2004/wd-wsdl20-patterns-20040326/.

[31] WSDL, Web Service Description Language. http://www.w3.org/tr/wsdl.

[32] A. Moormann Zaremski and J. M. Wing. Specification matching of software components. *ACM Trans. on Software Engineering and Methodology*, 6(4):333–369, 1997.

Matteo Baldoni, Cristina Baroglio, Alberto Martelli,
Viviana Patti, and Claudio Schifanella
Dipartimento di Informatica
Università degli Studi di Torino
C.so Svizzera, 185
I-10149 Torino (Italy)
e-mail: {baldoni,baroglio,mrt,patti,schi}@di.unito.it

Whitestein Series in Software Agent Technologies, 23–39

Enabling Business Experts to Discover Web Services for Business Process Automation

Sebastian Stein, Katja Barchewitz and Marwane El Kharbili

Abstract. Using Web services for business process automation is an accepted approach in context of service-oriented architectures (SOA). Business process models are created by business experts usually not having an IT background and who are therefore not able to use the technical descriptions available for web services. In this paper, we show how we extended the market leading business process management suite ARIS to enable business experts to discover, assess, and select Web services for business process automation. We developed a structural and a semantic matching algorithm as well as a graphical user interface for Web service assessment. We use a schema to classify Web service discovery literature and we relate our work to it. Our completely integrated discovery tool helps bridging the gap between business and IT, because business experts can now discover Web services needed for business process automation on their own.

Keywords. web service discovery, bpm, aris.

1. Introduction

A company is driven by its business processes and their interfaces to the outside world. Those business processes are documented using business process notations like Event-driven Process Chains (EPC) [1]. Such process models are used on different abstraction levels, for example to specify how a business activity should run in general but also how a certain sub-process should be implemented using information systems. It is the grand vision of business process management (BPM) [2] to directly derive the process implementation from the business process models created by business experts.

In past years, the idea of service-oriented architectures (SOA) [3] became a popular approach for integrating information systems to support business process automation. SOA itself is just an architectural style, but not a specific technology.

There seems to be a preliminary consensus in enterprise computing that Web service technology [4, see e. g.] is the preferred SOA implementation solution. Steps in a business process are automated by Web services and the business process models are afterwards transformed into process execution languages like the Business Process Execution Language (BPEL) [5].

Using Web services for business process automation has a major drawback: Business processes are described by business experts, whereas Web services are described on a technical level. The technical Web service description is not usable for business experts and therefore they are not able to select a web service to automate a certain step in a business process. To overcome this problem, we developed a tool and method allowing business experts to discover Web services for process automation. The discovery tool is completely integrated in the world market leading[1] software for business process analysis and management *ARIS*[2] and the belonging ARIS methodology [8] for business process management.

This article is structured as follows. In the next section we provide a detailed literature review of Web service discovery algorithms. Based on existing literature, we develop a classification for Web service discovery approaches. In section 3 we present our solution. First, we discuss in sub-section 3.1 the general structure of our solution and how our solution can be classified according to the classification schema developed. In sub-sections 3.2 and 3.3 we describe how Web services and data structures are represented in ARIS. This information is important in order to understand the description of the two matching algorithms. Our structural matching algorithm for Web service discovery is explained in sub-section 3.4. Our semantic matching algorithm for Web service discovery is explained in sub-section 3.5. In sub-section 3.6 we present the graphical user interface we developed to allow business experts to assess the matching results and to make an informed decision of the Web service to be used. In section 4 we give an example to better illustrate our solution. Finally, we present our conclusions at the end of the article.

2. Literature Review and Theoretical Foundation

We investigated service discovery literature. Even though we were not able to identify any specific publication dealing with Web service discovery for business process automation, we found many publications related to web service discovery in various domains. For example, many publications are targeting Web service discovery in context of Grid computing [9]. Here, services are bound during execution often based on quality of service (QoS) parameters. A related domain is agent systems [10, see e. g.] trying to identify a communication partner with a set of defined capabilities. Other publications deal with identifying Web services in context of software engineering. The idea is to construct complex (software) systems by combining basic Web services. Public market places are created so that

[1] . . . according to Gartner [6] and Forrester [7] market research reports. . .
[2] http://www.aris.com/

service providers can advertise their offerings and service consumers can evaluate and bind them. Today, public standards for such service registries are available like UDDI [11] and ebXML Registry Information Model [12], even though we cannot confirm a quick adoption of those standards in industry.

The idea of offering well encapsulated functionality to an anonymous market is not new. During the early 1990s the idea of component-oriented software engineering [13, see e. g.] became popular and here again it was the idea to reuse existing software components to construct more complex applications. However, there was no agreed standard for describing and binding the components and so their application was always limited to users with the same technology platform.

It can be said that Web service technology resolves this major interoperability issue by defining the WSDL [14] and SOAP [15] standards. The Web Service Description Language (WSDL) is used to define the interface of a Web service as well as where the Web service can be reached in terms of a unique resource identifier. SOAP defines a standard protocol to access the remote resource.

Web service discovery aims at identifying a Web service able to fulfil the requirements defined by the Web service request. We were able to identify three basic approaches to Web service discovery:

1. *Structural* discovery approaches use syntactical information available like the interface description and the definition of the data messages exchanged between the communication partners. This kind of matching is very technical, as it requires the service requester to specify structural requirements like a certain operation signature or data type. A typical example of such a discovery approach is given by Ramasamy [16]. Ramasamy compares operation names and operation parameters to the service request to discover Web services.
2. *Lexical* discovery approaches use natural language descriptions. For example, Web service operation names usually contain some terms describing their functionality. Also, WSDL and other standards allow embedding natural language descriptions. The lexical algorithms remove stop words from those descriptions, find synonyms using lexical databases like WordNet [17] and compute similarity coefficients. For example, Zhuang et al. [18] present an algorithm to compute the similarity of two web services. Their approach uses the information given in the WSDL files and does not require any additional annotations. They do manual pre-processing of the WSDL files to remove abbreviations, but it should be possible to use lexical databases like WordNet to automate this task in the future.
3. *Semantic* descriptions often based on ontologies are another major approach for Web service discovery. They use formal methods to describe web service capabilities and properties so that machine reasoning can be used to identify possible candidates for a service request. There are competing formalisms for describing this semantic information like the Web Service Modeling Ontology (WSMO) [19] or OWL-S [20]. A standard called WSDL-S [21] was proposed which provides some extensions for WSDL so that semantic descriptions in

any formalism can be referenced from a WSDL file and so semantic annotation of existing Web services becomes possible. An early example for semantic matching is Paolucci et al. [22]. They use DAML-S to describe the capabilities of a web service as well as the service request. In a more recent example Kritikos and Plexousakis [23] describe quality of service (QoS) parameters using OWL-S allowing matching on non-functional Web service properties.

Most discovery algorithms combine different approaches to achieve a better result. For example, Wang and Stroulia [24] combine structural and lexical analysis. Kokash et al. [25, p. 526] have identified several strategies how to combine the results of different discovery approaches.

- The *mixed* strategy uses different discovery approaches and matching algorithms in parallel and unites the returned result sets into one final result set. Normally, duplicates are removed from the final result set.
- The *cascading* strategy applies different discovery approaches and matching algorithms in sequence. A matching is only performed on the result set returned by the previous algorithm. This helps to reduce the amount of processing needed and it can increase the overall result quality. This can be seen as a stepwise refinement.
- The *switching* strategy selects between different discovery strategies and matching algorithms based on predefined criteria. For example, if the results returned by an algorithm are not satisfactory, another algorithm is used. The cascading and switching strategy can be combined to create more complex strategies.

Our experience and literature investigation show that there is another important characteristic to correctly classify Web service discovery approaches and matching algorithms. One has to distinguish between discovery during design time and run-time. The former is normally initiated by a user designing a web service composition for example to create a custom software application or to automate a business process. This is also sometimes referred to as early binding. The latter is used during execution of a service composition. In this case, the composition only contains a requirements definition for a service call but it does not specify which specific Web service to use. At run-time, discovery is done to find all Web services matching the requirements specification and the best fitting Web service is used. This is sometimes referred to as late binding.

According to Kokash et al. [25, p. 522]St Web service discovery consists of three major phases, also illustrated in figure 1:

1. During the *matching* phase matching algorithms belonging to the different discovery approaches are applied. The results are combined according to the chosen strategy. The result set might just consist of all Web services matching the request or they might be ranked according to their fitness.
2. During the *assessment* phase the matching results are further refined by a set of criteria. Where matching is normally done automatically, the assessment is

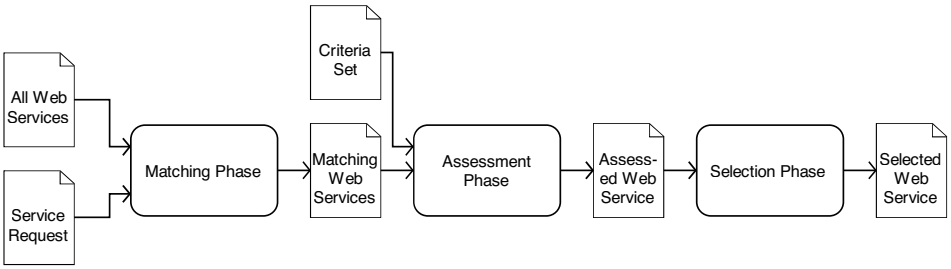


FIGURE 1. Major Phases of Web Service Discovery

often done manually, especially if Web service discovery is done during design time.

3. In the final *selection* phase a Web service is chosen and used in the composition as intended. This might also mean to adapt either the web service or the consuming process or application.

In this section, we provided an overview of current Web service discovery approaches and how to classify them. In the following section we show how our work relates to and extends existing approaches for Web service discovery.

3. Our Approach to Web Service Discovery

3.1. Overview

In theory it might be possible to discover a Web service during run-time to automate a certain step in a business process. Still, we have not seen something like that nor did our customers asked for it. They carefully design their processes and select Web services during design time. Therefore, our approach focuses on Web service discovery during design time.

We do not use any algorithms for lexical matching of Web services, even though the user can refine the matching results during the assessment phase using ordinary string search. In contrast, we make heavy use of structural matching to identify Web services able to handle the data objects modelled in the business process. We compare the business objects given in the business process to the message types used by the Web service in the message exchange. In that sense our matching algorithm is very similar to what Ramasamy [16] describes. We also do a lightweight variant of semantic matching. The users of our tool are able to create a taxonomy and use this taxonomy to classify the functionality of Web services. Even though we are not using ontologies or reasoning algorithms, it is still a way of capturing semantics.

We use the mixed strategy to unite the results of structural and semantic matching. We consider a Web service to fulfil the service request, if it is either discovered by structural or semantic matching or by both approaches. We remove any duplicates from the final result set before it is presented to the user for assessment.

We have structured the Web service discovery tool according to the three phases of service discovery. The user initiates Web service discovery by selecting the business process step to be automated. During the first phase, we analyse the context of the selected business process step and derive the service request. All Web services available in our tool are matched against the service request. Afterwards, the result set is presented to the user for assessment. Finally, the user selects the Web service to use and the Web service is automatically added to the business process.

The following sub-sections describe our solution in detail. We do not describe the user roles involved using this solution to make the description not too complicated. An example is given in section 4. This example provides a walk-through also describing the involved user roles.

3.2. Web Service Representation in ARIS SOA Architect

The ARIS Platform is a set of integrated products to manage all aspects of an enterprise model. Besides defining and documenting a business strategy and business processes, one important aspect of an enterprise is the supporting IT infrastructure. Today, many companies are migrating their IT infrastructure to service-oriented architectures. A common piece in such an architecture are web services. Therefore, the specific SOA related ARIS product called ARIS SOA Architect allows importing Web services, if they are described using the Web Service Description Language (WSDL) version 1.1. Instead of just dumping the file in the underlying database, we extract the content and represent it using the Unified Modelling Language (UML). For example, WSDL porttypes are mapped to UML interfaces and the belonging operations to UML operations. The WSDL import functionality of ARIS SOA Architect also allows importing embedded or referenced XML schema definitions. Those definitions are mapped to UML as well.

Using UML models to visualise the information contained in a WSDL file is a proven approach for technical oriented users, but it is insufficient for business users. Therefore, we also create an object representing the web service from a business perspective. This object has no technical information like operations, interfaces or technical message types, because a business user should not have to deal with this kind of information in order to use a web service. Instead, the Web service is described from a business perspective by adding tags to it. The tag concept is described in detail in section 3.5. Other information includes the hardware the Web service is running on, the application system the Web service belongs to, and the person responsible for the Web service. The user can also evaluate, who or which process is currently using the Web service and the company locations the Web service is available for.

3.3. Information Architecture and Business Objects in ARIS SOA Architect

As described in the previous sub-section, technical data structures like XML schema definitions are mapped to UML models. However, for a business user it is not useful to deal with such a detailed data model. For example, there might

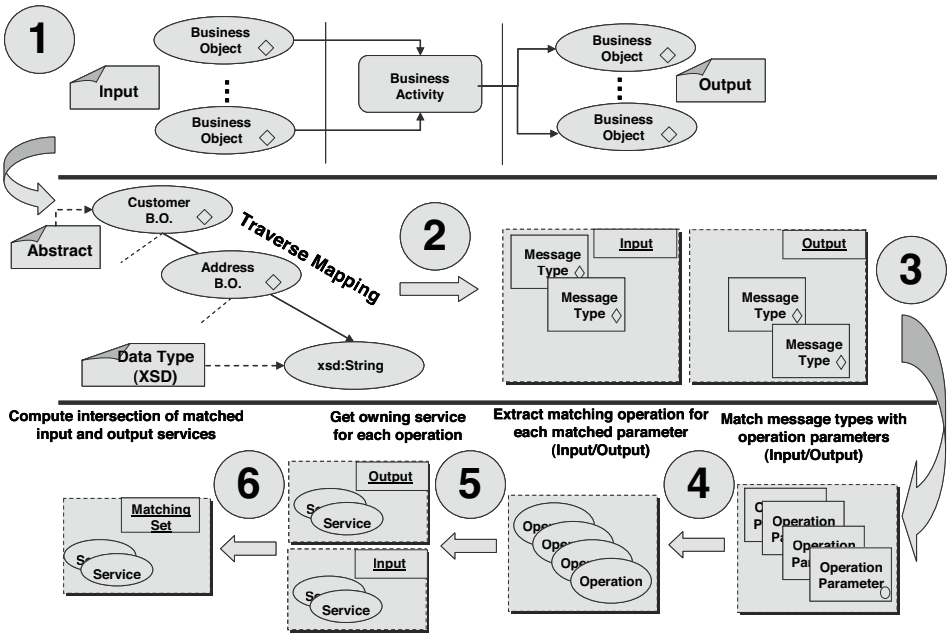


FIGURE 2. Structural Web Service Matching Algorithm

be different technical message types or database schemas to represent customer data, but from a business perspective there is only one customer data object. Such data objects are often called business objects or logical data objects. As in case of technical data modelling, business objects are also further refined into more concrete parts. For example, the business object customer can be decomposed into the name, address, payment history, and an interest profile. The models describing all relevant business objects for the whole enterprise are called information architecture. An internationally operating company should only have one information architecture, but there are usually several implementation of this architecture.

Message types defined by Web services are an implementation of business objects, too. A business user is using business objects to specify the data flow in a business process. If the technical data objects defined by the web services are mapped to the business objects used by the business user, it is possible to discover Web services for business process automation. The underlying algorithm is explained in detail in the following sub-section.

3.4. Structural Matching Algorithm

Web services use message types to define their input and output. On the other hand, a business process uses business objects to describe the data flow. Both concepts are not equivalent, because they are on completely different abstraction levels as explained in the previous section. Instead of using message types in business

processes to model the data flow, one should create a mapping between business objects and message types. This mapping can be used to discover Web services by navigating from the business objects over the message types to the belonging Web services.

Our structural matching algorithm works as follows: We first extract all business objects modelled as input and output of the business process step as illustrated in step 1 in figure 2. Afterwards, we have two sets, one containing all business objects required as input and the other one containing all business objects required as output. For each of those business object sets we navigate through the mapping to identify all message types. Implementing this navigation is not trivial, because the modelling capabilities of the ARIS suite allows as many abstraction levels between business object and message type as the user wants including cyclic dependencies. Optimisation techniques must be used to implement a high performing solution. For example, the business object customer used in a business process is further decomposed into an address. This address can be represented using different message types. The algorithm has to identify all message types mapped to the business object. This is illustrated between steps 1 and 2 in figure 2. After this step, we have two sets of message types, one for message types required as input and one for message types required as output. In step 3 we check to which operation parameters those message types belong and if the parameters have the same direction as the message types (input or output). Extracting this information is possible, because we map the complete content of the WSDL file and related XSD files to UML models as described in section 3.2 and 3.3. If the message type is an operation parameter with the correct direction, we extract the belonging operation as shown in step 4. Afterwards, the operation's owning Web service is extracted in step 5. At the end we have two sets of Web services: one set supporting all input business objects and the other set containing all web services supporting the output business objects. In the final step 6 both result sets are intersected. The final result set of the structural matching algorithm contains only those Web services, which are part of both preliminary result sets and are therefore able to support all input as well as all output business objects.

As one can see, we do not check that a Web service has at least one single operation able to support all business objects in the parameter list. While automating business processes, this is normally not a problem, because during transformation of a business process into an executable process model (like BPEL), a process step can be split up into several technical steps. Also, adding another operation to a Web service able to handle all business objects in one request is often possible, if the Web service is owned by the company.

The biggest disadvantage of the structural matching algorithm is the effort required for mapping business objects to technical data structures. Many of our customers have already created a comprehensive information architecture consisting of the most important business objects, but matching those business objects to

technical data structures requires effort. Each customer must decide, if this investment can be justified. As an alternative, we provide a more lightweight matching algorithm, which is described in the following sub-section.

3.5. Semantic Matching Algorithm

Not all customers are interested in creating and managing an information architecture. Therefore, we provide a second more lightweight approach for web service discovery. First, the user creates a taxonomy for functional descriptions. Each taxonomy object has a very short textual description, comparable to a tag. In addition, a more detailed description including texts and diagrams can be added so that the meaning of the tag can be illustrated for human users.

The taxonomy is used to annotate Web services by assigning the tags to them. Tags should be shared between Web services, if Web services have similar properties or functional capabilities. For example, the tag *web interface* should be added to all Web services providing a web based user interface.

The taxonomy must be carefully designed and managed. For example, not every user should be able to extend the taxonomy by creating new tags. Instead, reuse of existing tags must be enforced. Tags must also be described in a way that users have a clear understanding of their meaning.

The taxonomy is also used during business process modelling to express what kind of functionality is needed to automate a business process step. Tags are assigned to a business process step for this purpose.

The semantic matching algorithm first extracts all tags assigned to the business process step to be automated. The extracted tags describe the service request. Afterwards, we extract the tags assigned to each Web service and compare this list to the service request. This way we can discover those web services able to support the service request. This algorithm is much simpler compared to the structural matching algorithm. For example, the algorithm does not support decomposition of tags, so a Web service will not be discovered, if it has only a more general tag assigned as specified in the service request. We do not see this as a drawback, because this discovery algorithm is meant to be lightweight and easy to understand.

3.6. Web Service Assessment and Refinement

The final result set consists of all Web services discovered either by the structural matching algorithm or by the semantic matching algorithm. The matching results are shown to the user in a graphical user interface. A screenshot can be seen in figure 3. The screen design consists of three parts, which are marked in the screenshot with the numbers 1–3.

During the assessment phase the user further refines the result set. For example, the user can search the descriptions and names of the Web services with a string search or he can filter the list of Web services according to their namespace. He can also filter the list according to the date the web services were imported

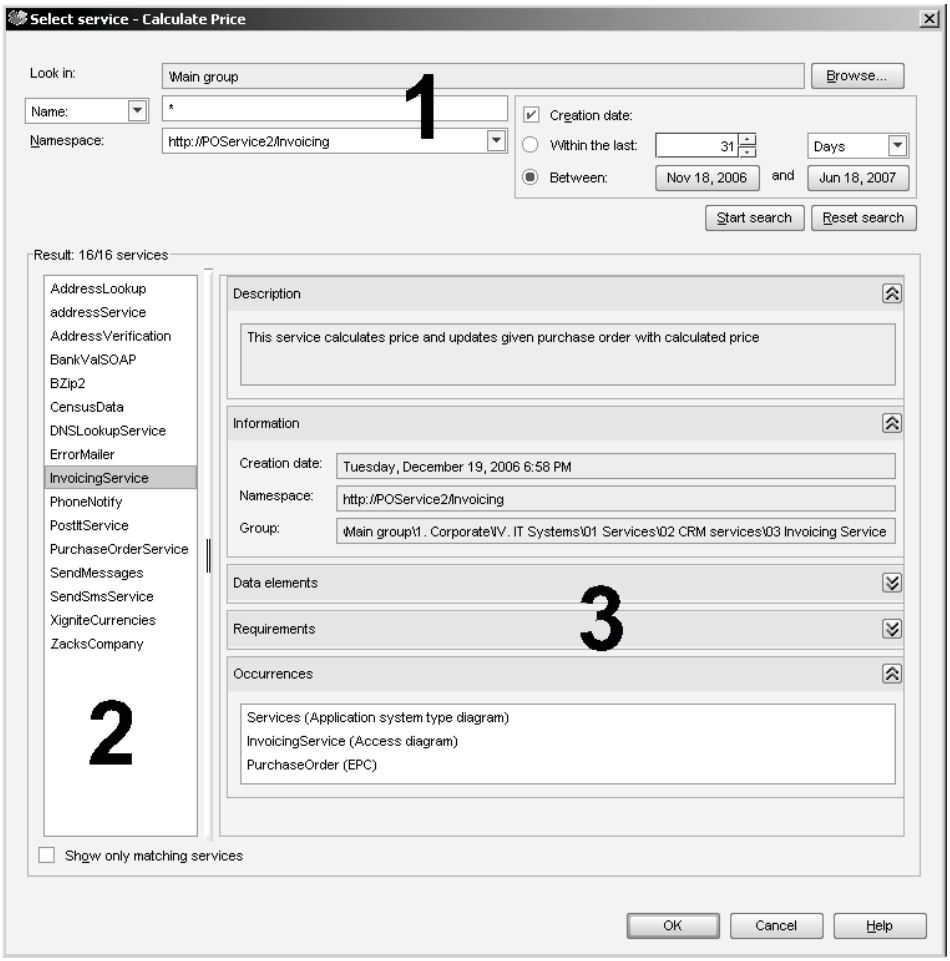


FIGURE 3. Graphical User Interface of the Web Service Discovery Tool in ARIS SOA Architect

into ARIS. Those refinement settings are done in part 1 of the screen design as shown in figure 3.

The current result set can be seen in part 2 of the screen design. This part also allows switching between a list of all Web services and the list with matching Web services. This is useful in case the matching algorithms did not return a satisfying discovery result.

The user has to assess if a Web service fulfils the service request. This assessment cannot be done based on the name of a Web service. Therefore, additional information is shown in part 3 of the screen design for the currently selected Web service. For example, all business objects supported by the Web service are shown

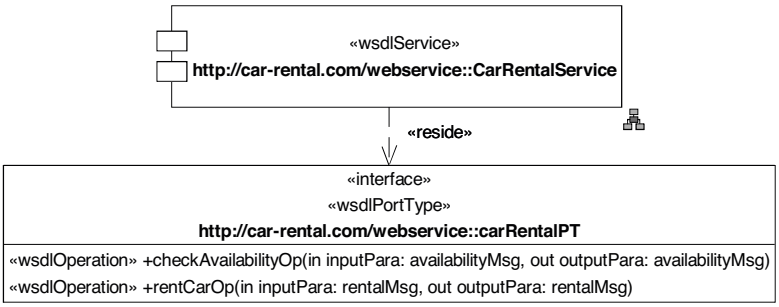


FIGURE 4. Web Service as UML Component Diagram

as well as the textual description. It is also possible to see in which other contexts the Web service is used.

Finally, the user selects a Web service and confirms this selection. The dialog closes and the Web service is automatically attached to the business process step. Now that a Web service is assigned to the business process step, this process step is automated from a design point of view. If all steps in a business process are supported by Web services, the model can be transformed to BPEL as we have shown in [26].

4. Example

This section provides an example to better illustrate our discovery approach. The example mentions two different roles – a business analyst and an IT architect. The business analyst has no IT background, but instead experience in business process modelling. The IT architect has SOA know-how and is able to use typical SOA middleware products and standards. In reality, there are usually more roles involved, but we tried to make the example not too complicated.

A fictitious company defines an internal business process for organising business trips. If such a business trip has to be done by car, the employee has to use a company car, if available. Only if no company car is available, the employee is allowed to rent a car from a defined car rental company.

The car rental company provides a Web service for this purpose. In order to be able to use this Web service in business process modelling, the Web service must be made available in ARIS. An IT architect imports the Web service. The content of the WSDL file is visualised as an UML component diagram as shown in figure 4.

Besides using this technical information, the IT architect annotates the web service with tags from the company wide taxonomy to describe the service semantically. The company wide taxonomy is defined prior and will not be changed by the IT architect. In addition, the IT architect might add who is responsible for

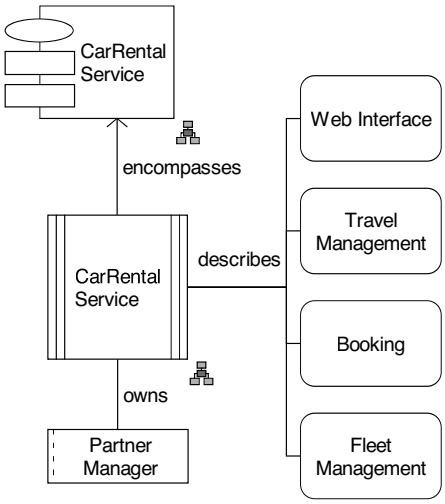


FIGURE 5. Annotated Web Service

this Web service. Figure 5 shows the annotated web service. There are four tags describing the Web service and an owner is defined, as well.

If the company has an information architecture, the IT architect maps the message types used by the Web service to the belonging business objects. This can be a complex task and he might have to consult business analysts to identify the correct business objects. The mapping is done in different diagrams, which are not shown.

A business analyst models the business process described at the beginning of this section. Figure 6 shows a small part of the business process. The business analyst creates a business function and connects it with the input and output business objects. In addition, the business expert specifies requirements by relating the business function to tags from the company wide taxonomy. In reality, companies have either an information architecture or a company wide taxonomy, but not both.

The business analyst wants to automate the business function using a web service. He selects the business function and starts the integrated discovery tool. The discovery tool evaluates the content of the business process by extracting all input and output business objects and extracting the tags connected to the business function. This information is the input for the semantic and structural matching algorithms as described in section 3. The results are shown to the business analyst in the graphical user interface discussed in section 3.6 and shown in figure 3. The business analyst selects a Web service after assessing the different choices. The Web service is automatically added to the business process as shown in figure 7. The symbol of the business function is changed as well to visualise that this process step is now automated by a Web service.

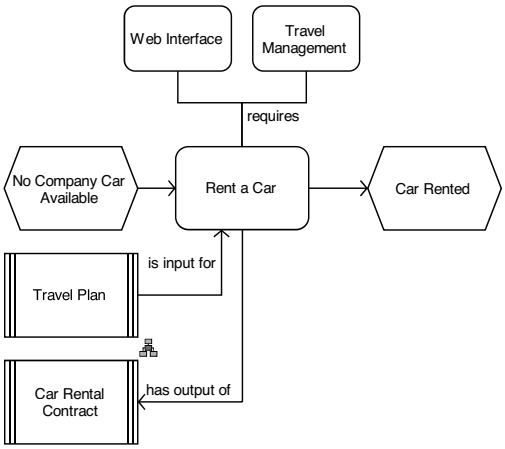


FIGURE 6. Business Process without Web Service

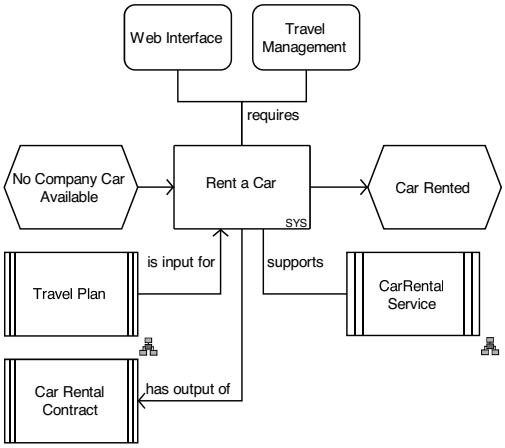


FIGURE 7. Business Process with Web Service

The resulting business process cannot be executed directly, because different technical information is missing. An IT analyst uses our EPC to BPEL transformation [26] to generate a corresponding BPEL model. This BPEL model must be further refined, for example selecting correct Web service operations or defining technical exception handling.

The example given in this section shows that a business analyst is able to automate business processes by discovering matching Web services. In order to select a Web service the business analyst does not need IT knowledge. On the other hand, an IT expert implementing a business process gets a detailed specification for his work.

5. Conclusions

In this article we presented a Web service discovery tool for business process automation. The tool is completely integrated in the world market leading tool for business process management ARIS and the belonging ARIS method. In contrast to other publications, our approach clearly separates between the different abstraction levels by not mixing technical data with technology independent business processes. The Web service discovery tool is structured around the three discovery phases: matching, assessment, and selection. The intended audience are non technical users like business analysts. The tool does not confront them with unnecessary technical details. The matching algorithms used discover a set of matching services. The result set can be assessed and refined by the user or the user can switch to a list with all available Web services if needed. For each Web service we provide additional information so that the user can make an informed decision while selecting a Web service.

In order to discover Web services, we use a structural matching algorithm and a semantic matching algorithm. In both cases, the service request is extracted from the business process model. No user input is required for defining the service request, which simplifies the overall tool usage. The structural matching algorithm identifies all Web services able to support the business objects modelled at the business process step to be automated. This is possible based on a mapping of Web service message types to business objects. The semantic matching algorithm requires that Web services and the business process step are tagged using a taxonomy. A Web service is considered to match semantically, if it has at least all tags also attached to the business process step.

So far, the tool was already deployed by several customers. Many technical oriented users were fascinated by the structural matching algorithm, but it seems that business oriented users like the semantic matching algorithm more. However, at the current point we have not received enough feedback from the field to make any final judgement. We already initiated a survey among the first users, but the results are not available yet.

Even though our customers perceive our current solution for Web service discovery as good, there is still room for improvements. For example, the structural matching algorithm does not scale very well, because we are not able to use any optimisation techniques like pre-indexing or caching. This is not a problem inherent in our algorithms, but related to the technological framework we have to use. It is our challenge for the next months to find optimisation tricks to overcome those problems. Another point of improvement is to allow a more sophisticated semantic matching. For example, it should be possible to create a tag hierarchy so that Web services are discovered, even if a more general tag was assigned to them. Also, it should be possible to express that two tags cannot be used together, because they contradict each other. However, we do not intend to provide complete ontology modelling and matching possibilities in the near future, because the tool must be easy and intuitive to use even without any special training. We also plan

to extend our web service discovery concept to a more general service discovery concept. Basically, the service concept can be used to describe any kind of business function. We will broaden the definition of the service concept so that it better aligns with the service concept as defined in OASIS' SOA Reference Model [3]. For example, a service must not be implemented using software at all. We will extend our discovery approach to cover such business services as well.

Our most important contribution is to bring Web service discovery to a non-technical audience. Web service discovery can now be done by business analysts. Even though there is still room for improvements, we are confident that our tool helps bridging the gap between business and IT.

6. Acknowledgement

A first prototype of the discovery tool was partly funded by the German federal ministry of education and research within the public research project OrViA (http://www.orvia.de/). The literature review as well as preparing this paper was supported by the EU Commission within the integrated research project SUPER (http://www.ip-super.org/). We like to thank the German federal ministry of education and research and the EU Commission for this opportunity!

References

[1] Scheer, A.W., Thomas, O., Adam, O.: Process modelling using event-driven process chains. In Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., eds.: Process-Aware Information Systems. Wiley, Hoboken, New Jersey, USA (2005) 119–146

[2] Smith, H., Fingar, P.: Business Process Management: The Third Wave. 1st edn. Meghan-Kiffer Press, Tampa, FL, USA (2003)

[3] MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R.: Reference model for service oriented architecture 1.0. Technical report, OASIS (July 2006) http://www.oasis-open.org/committees/download.php/19361/soa-rm-cs.pdf.

[4] McGovern, J., Sims, O., Jain, A., Little, M.: Enterprise Service Oriented Architectures. Springer, Dordrecht, The Netherlands (2006)

[5] Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guizar, A., Kartha, N., Liu, C.K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web services business process execution language (bpel) version 2.0. Technical report, OASIS (April 2007)

[6] Blechar, M.: Magic quadrant for business process analysis market, 2h07. Technical report, Gartner (June 2007)

[7] Peyret, H.: The forrester wave: Enterprise architecture tools, q2. Technical report, Forrester (April 2007)

[8] Scheer, A.W.: ARIS - Business Process Frameworks. 3rd edn. Springer, Berlin, Germany (1999)

[9] Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications **15**(3) (2001) 200–223

[10] Weiss, G.: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press (1999)

[11] Clement, L., Hately, A., von Riegen, C., Rogers, T.: Uddi version 3.0.2. Technical report, OASIS (October 2004) http://www.oasis-open.org/committees/uddi-spec/.

[12] Fuger, S., Najmi, F., Stojanovic, N.: ebxml registry information model version 3.0. Technical report, OASIS (May 2005) http://docs.oasis-open.org/regrep-rim/v3.0/.

[13] Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley (1997)

[14] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web service description language (wsdl) 1.1. Technical report, W3 Consortium (March 2001) http://www.w3.org/TR/wsdl.

[15] Mitra, N., Lafon, Y.: Soap version 1.2. Technical report, W3 Consortium (April 2007) http://www.w3.org/TR/soap.

[16] Ramasamy, V.: Syntactical & semantical web services discovery and composition. In: The 8th IEEE International Conference on E-Commerce Technology and the 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE'06). (2006)

[17] Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press (1998)

[18] Zhuang, Z., Mitra, P., Jaiswal, A.: Corpus-based web services matchmaking. In: Workshop on Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing, held in conjunction with The Twentieth National Conference on Artificial Intelligence (AAAI '05), Pittsburgh, PA, USA (July 2005)

[19] Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer (2006)

[20] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: Owl-s: Semantic markup for web services. Technical report (2004) http://www.daml.org/services/owl-s/.

[21] Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.T., Sheth, A., Verma, K.: Web service semantics (wsdl-s) version 1.0. Technical report, W3 Consortium (November 2005) http://www.w3.org/Submission/WSDL-S/.

[22] Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: The Semantic Web - ISWC 2002: First International Semantic Web Conference. LNCS 2342/2002, Sardinia, Italy, Springer, Germany (June 2002)

[23] Kritikos, K., Plexousakis, D.: Semantic qos metric matching. In: 4th European Conference on Web Services (ECOWS). (December 2006) 265–274

[24] Wang, Y., Stroulia, E.: Flexible interface matching for web-service discovery. In: Web Information Systems Engineering (WISE). Proceedings of the Fourth International Conference on. (2003) 147–156

[25] Kokash, N., van den Heuvel, W.J., D'Andrea, V.: Leveraging web services discovery with customizable hybrid matching. In Dan, A., Lamersdorf, W., eds.: Service-Oriented Computing (ICSOC 2006). Proceedings of the Fourth International Conference on. LNCS 4294, Berlin, Germany, Springer (2006) 522–528

[26] Stein, S., Ivanov, K.: EPK nach BPEL Transformation als Voraussetzung für praktische Umsetzung einer SOA. In Bleek, W.G., Raasch, J., Züllighoven, H., eds.: Software Engineering 2007. Volume 105 of Lecture Notes in Informatics (LNI)., Hamburg, Germany, Gesellschaft für Informatik (GI) (March 2007) 75–80

Sebastian Stein
IDS Scheer AG
ARIS Research
Altenkesseler Str. 17
66115 Saarbrücken
Germany
e-mail: sebastian.stein@ids-scheer.com

Katja Barchewitz
IDS Scheer AG
Altenkesseler Str. 17
66115 Saarbrücken
Germany
e-mail: katja.barchewitz@ids-scheer.com

Marwane El Kharbili
IDS Scheer AG
ARIS Research
Altenkesseler Str. 17
66115 Saarbrücken
Germany
e-mail: marwane.elkharbili@ids-scheer.com

Whitestein Series in Software Agent Technologies, 41–58

Evaluation of Semantic Service Discovery - A Survey and Directions for Future Research

Ulrich Küster, Holger Lausen and Birgitta König-Ries

Abstract. In recent years a huge amount of effort and money has been invested in the area of semantic service discovery and presented approaches have become more sophisticated and mature. Nevertheless surprisingly little effort is being put into the evaluation of these approaches. We argue that the lack of established and theoretically well-founded methodologies and test beds for comparative evaluation of semantic service discovery is a major blocker of the advancement of the field. To lay the ground for a comprehensive treatment of this problem we discuss the applicability of well-known evaluation methodologies from information retrieval and provide an exhaustive survey of the current evaluation approaches.

Keywords. semantic web services, service discovery, evaluation.

1. Introduction

In recent years semantic services research has emerged as an application of the ideas of the semantic web to the service oriented computing paradigm. Semantic web services (SWS in the following) have received a significant amount of attention and research spending since their beginnings roughly six years ago [8]. Within the sixth EU framework program[1] (which ran from 2002 to 2006) alone at least 20 projects with a combined funding of more than 70 million Euro deal directly with semantic services which gives a good impression of the importance being currently put on this field of research. In the following we will focus on efforts in the field of SWS discovery and matchmaking. We refer to discovery as the whole process of retrieving services that are able to fulfill a need of a client and to matchmaking as the problem to automatically match semantically annotated service offers with a semantically described service request. However, we think that our findings also apply to other related areas, like automated semantic service composition.

[1] http://cordis.europa.eu/fp6/projects.htm

In this paper we argue that despite of the huge amount of effort (and money) spent into SWS discovery research and despite the fact that the presented approaches become more sophisticated and mature, much too little effort is put into the evaluation of the various approaches. Even though a variety of different service matchmakers have been proposed we did not suceed to find any publications with a thorough, systematic, objective and well designed evaluation of those matchmakers. This corresponds to a trend that seems to exist in computer science in general. In [14] Tichy et. al. find that computer scientists publish relatively few papers with experimentally validated results compared to other sciences. In a follow-up work [13] Tichy claims that this trend is harmful for the progress of the science. There are positive examples that back his claim:

> "[in the experiments] ... there have been two missing elements. First [...] there has been no concerted effort by groups to work with the same data, use the same evaluation techniques, and generally compare results across systems. The importance of this is not to show any system to be superior, but to allow comparison across a very wide variety of techniques, much wider than only one research group would tackle. [...] The second missing element, which has become critical [...] is the lack of a realistically-sized test collection. Evaluation using the small collections currently available may not reflect performance of systems in large [...] and certainly does not demonstrate any proven abilities of these systems to operate in real-world [...] environments. This is a major barrier to the transfer of these laboratory systems into the commercial world."

This quote by Donna Harman [5] addressed the situation in text retrieval research prior to the establishment of the series of TREC conferences[2] in 1992 but seems to perfectly describe the current situation in SWS discovery research. Harman continued:

> "The overall goal of the Text REtrieval Conference (TREC) was to address these two missing elements. It is hoped that by providing a very large test collection, and encouraging interaction with other groups in a friendly evaluation forum, a new thrust in information retrieval will occur."

From the perspective of today, it is clear that her hope regarding the positive influence of the availability of mature evaluation methods to the progress of information retrieval research was well justified. In this paper we argue that a similar effort for SWS related research is necessary today for the advancement of this field.

The rest of this paper is organized as follows. In Section 2 we will review the philosophy of information retrieval evaluation and argue that traditional evaluation methods can not be applied easily to the domain of SWS matchmaking. In Section 3 we provide an extensive survey of current evaluation efforts in the area of SWS discovery. We cover the related work in Section 4, draw conclusions about what

[2] http://trec.nist.gov/

is missing so far and provide directions for future work in Section 5 and finally summarize in Section 6.

2. The Philosophy of Information Retrieval Evaluation

In a broader context, discovery of semantic web services can be seen as a special information retrieval (IR) problem. According to Voorhees [17], IR evaluation has been dominated for four decades by the Cranfield paradigm which is characterized by the following properties:

- An IR system is mainly evaluated by means of *recall* and *precision*.
- Recall is defined as the proportion of retrieved documents that are *relevant*.
- Precision is defined as the proportion of *relevant* documents that are retrieved.
- Relevance is based on topical similarity as obtained from the judgements of domain experts.
- Test collections therefore have three components: a set of documents (the test data), a set of information needs (topics or queries) and a set of relevance judgements (list of documents which should be retrieved)

Vorhees identifies several assumptions on which the Cranfield paradigm is based that are unrealistic in most cases. She concludes that experiments based on those assumptions are a noisy process but is able to provide evidence that – despite of the noise – such experiments yield useful results, as long as they are only used to assess the *relative performance* of different systems evaluated by the *same experiment*.

Since the experiments based on the Cranfield paradigm are extremely well established, since the methodology of these experiments is well understood and since SWS matchmaking is a special information retrieval problem, it seems obvious to try to apply the same methods and measures to the SWS matchmaking domain. However, in the following we argue that this is not a promising approach for various reasons.

A model of the general process of information retrieval is depicted in Figure 1. The user has a real world need (like information about a certain topic) that needs to be satisfied with the existing real world supply (like a collection of documents). Both the need and the supply are abstracted to a model. In the case of web search engines for instance, such a model will consist of descriptors extracted from the query string and data structures like indexes built upon descriptors extracted from the web pages etc. The information retrieval system then operates on that model to match the need with the supply and returns the (real world) results. As a matter of fact the power of this model (how well it captures the real world and how well it supports the retrieval, i.e. matchmaking and ranking process) is of critical importance for the retrieval system and thus a central component of its overall performance.

Traditional information retrieval systems typically create the model they operate on in an autonomous fashion. Thus, from the viewpoint of an evaluation

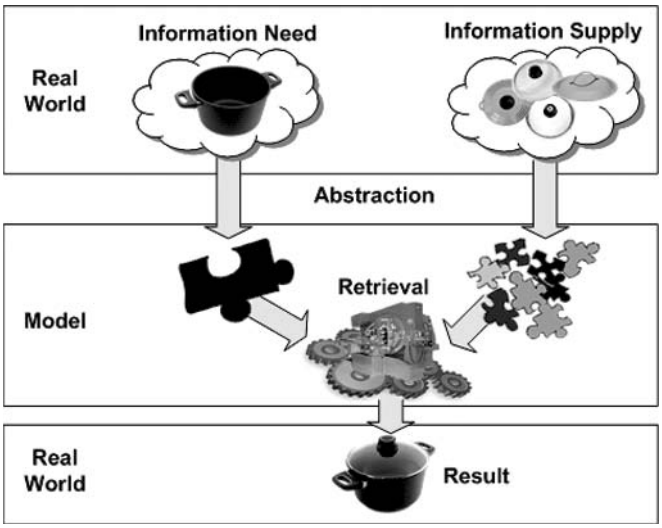


FIGURE 1. The process of information retrieval

they operate on the original data. Consequently, completely different IR systems can be evaluated on a common test data set (like a collection of documents).

SWS matchmaking follows a different paradigm. Here the semantic annotation is the model that is exploited during the matchmaking and it is not created automatically, but written by human experts. Currently there is no agreed upon formalism used for the semantic annotations, but competing and mostly incompatible formalisms are in use (like WSMO[3], OWL-S[4], WSDL-S[5], DSD[6], ...).

To apply the Cranfield paradigm to the evaluation of SWS discovery, one could provide a test collection of services in a particular formalism (e.g. OWL-S) and limit participation in the experiment to systems based on that formalism. This is the approach of the current S3 Matchmaker Contest (Section 3.1). Unfortunately, this excludes the majority of systems from participation. But there is an even more severe issue. Virtually all semantic matchmaking systems that are based on some form of logical reasoning operate deterministically. Therefore the question whether a *semantic* offer description matches a *semantic* request description in the same formalism can usually be decided unambiguously yielding perfect recall and precision (only depending on the definition of *match*). The task to evaluate, however, is the retrieval of *real-world* services that match a *real-world* request. Thus, the major source of differentiation between various approaches is

[3]http://www.wsmo.org/
[4]http://www.daml.org/services/owl-s/
[5]http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/
[6]http://hnsp.inf-bb.uni-jena.de/DIANE/

the expressivity of the employed formalism and reasoning. The critical questions here are:

- How precisely can a description based on a particular formalism reflect the real-world semantics of a given service (offer or request)?
- How much of the information contained in the descriptions of a service can a matchmaker use efficiently and effectively to decide whether two services match?

Note that the first question usually calls for more expressive formalisms whereas the second one requires less expressive formalisms to keep the reasoning tractable.

As argued above, the formalism employed for the semantic annotation of the services, i.e. the model used for the matchmaking, is of crucial importance for the overall performance of the discovery system. Consequently, a good evaluation of SWS discovery should measure not only the performance of a system for a given formalism, but also evaluate the pros and cons of that formalism itself. In fact, as long as there is no common understanding about the pros and cons of different formalisms and no agreement about which formalism to employ for a given task, evaluation of SWS discovery should first and foremost help to establish this missing common understanding and agreement.

The approach outlined above, however, neglects the influence of the model used in the matchmaking process and therefore does not measure the performance of that part of the retrieval process, which has the largest influence to the overall performance of the system.

A different approach that overcomes this limitation would be to provide a test collection of services in any format (e.g. human language) and let the participants annotate the services manually with their particular formalism. This is the approach taken by the SWS-Challenge (Section 3.2) and the DIANE evaluation (Section 3.3)). Unfortunately there are problems with this proceeding, too. First, such an experiment can hardly be performed on a large test collection, since the effort for the participants to manually translate the services into their particular formalisms is enormous. Yet, the unavoidable noise of experiments based on the Cranfield paradigm precisely requires large test collections to yield stable results [17]. Second, due to the human involvement, such an experiment can not be conducted in an automated way. Even worse, such an experiment does not only measure the performance of the matchmaking formalism and system, but also the abilities of the experts that create the semantic annotations. This introduces a whole new dimension of noise to the evaluation.

For the reasons given above we conclude that the experimental setup and the evaluation measures and methods developed for traditional information retrieval do not transfer directly to the SWS discovery domain. The influence of the described problems needs to be explored and new methods and measures have to be developed where necessary. To lay the foundation for this task, we provide an extensive survey of the current efforts in SWS discovery evaluation in the following section.

3. A Survey Of Current Approaches in Semantic Web Service Discovery Evaluation

3.1. S3 Matchmaker Contest

Klusch et al. have recently announced an annual international contest S3 on Semantic Service Selection[7] whose first edition will be held in conjunction with the upcoming International Semantic Web Conference in Busan, Korea (November 2007). We would like to express our acknowledgement and appreciation of this new effort that we welcome very much. Despite of that we identify some problems in the current setup of the contest.

The contest is based on a test collection of OWL-S services and "evaluation of semantic web service matchmakers will base on classic performance metrics recall/precision, F1, average query response time"[7]. As argued in the previous section this style of application of the Cranfield paradigm to the SWS matchmaking domain has a limited scope and significance since it does not allow a comparative evaluation of different semantic formalisms.

We furthermore think there is currently a problematic flaw in the practical setup of the contest, too. The most severe achilles' heel of any such contest is the dependency on a good SWS test collection. This year the S3 contest will rely solely upon the OWL-S Test Collection 2[8] which we believe to be unsuitable for a meaningful comparative and objective SWS matchmaking evaluation. We will explain our skepticism by a critical review of the collection.

The OWL-S Test Collection 2[9] is the only publicly available test collection of semantically annotated services of mentionable size. It has been developed within the SCALLOPS project[10] at the German Research Centre for Artificial Intelligence (DFKI). The most recent version 2.1 of the collection (OWLS-TC2 released in October 2006) contains 582 semantic web services written in OWLS 1.1. To put our following criticism into the correct light and in acknowledgement that currently no better public standard test collection exists, we would like to mention that the OWLS-TC2 does not claim to be more than "one possible starting point for any activity towards achieving such a standard collection by the community as a whole"[6]. Our criticism of OWLS-TC2 covers three aspects.

Use of realistic real-world examples. One common criticism to many use cases and evaluations in the service matchmaking domain is the use of artificial toy examples which are far from realistic applications. Even though examples do not necessarily have to be realistic to test features of a matchmaking system, the use of real-world examples clearly minimizes the danger of failing to detect lacking features

[7] http://www-ags.dfki.uni-sb.de/~klusch/s3/

[8] It is planned to extend the scope of the contest beyond OWL-S based matchmakers in the future. However, public test collections based on other formalisms have unfortunately not been developed so far. The S3 contest organizers have set up a public wiki (http://www-ags.dfki.uni-sb.de/swstc-wiki) to initiate efforts in this direction.

[9] http://projects.semwebcentral.org/projects/owls-tc

[10] http://www-ags.dfki.uni-sb.de/~klusch/scallops/

or awkward modeling. Furthermore, toy examples far from real-world applications critically hinder the acceptance of new technology by industry. OWLS-TC2 claims that "the majority of [...] services were retrieved from public IBM UDDI registries, and semi-automatically transformed from WSDL to OWL-S"[6]. Thus, one would expect somewhat realistic services but a substantial share of the 582 services of OWLS-TC2 seems quite artificial and idiosyncratic. Oftentimes the semantic of the service is incomprehensible even for a human expert and unfortunately only six of the original WSDL files are included in the test set download. A comprehensive coverage is impossible due to the size of OWLS-TC2 but the following examples illustrate the issues (in the following service names always refer to the name of the corresponding service description file, not the service name from the service's profile, quotes are from the service's description file or the OWLS-TC2 manual):

- Some services are simply erranous, quite a few services for instance are pairwise identical except for the informal textual description (e.g. `_price_CannonCameraservice.owls` and `_price_Fishservice.owls`)
- The service `__destination_MyOfficeservice.owls` is supposed to "return destination of my office", but takes concepts of type *organization* and *surfing* (which is a subclass of *sports*) as input.
- The service `surfing_farmland_service.owls` is described as "This is the recommended service to know about the farmland for surfing" and has an input of type *surfing* and an output of type *farmland*. What's the semantic of this service?
- The service `qualitymaxprice_cola_service.owls` "provides a cola for the maximum price and quality. The quality is an optional input." It is described by its inputs of type *maxprice* and *quality* and an output of type *cola*. There are a whole lot of similar services that return cola (six more services), beer + cola, coffee + whiskey (eleven services), cola-beer, cola + bread or biscuit (two services), drinks (three services), liquid, whiskey + cola-beer as well as irish coffee + cola. It remains unclear what is the semantics of these services.

Besides these issues we believe that examples from domains like *funding of ballistic missiles*, which the typical user of an evaluation system does not have any experience with, make a realistic evaluation unnecessary difficult.

Semantic richness of descriptions. Services should not only be realistic and realistically complex, they also should be described in sufficient detail to allow for meaningful semantic discovery. After all there should be an advantage to use semantic annotations compared to simply using traditional information retrieval techniques. Unfortunately the services of OWLS-TC2 are described extremely superficial. First of all it seems that all services are solely described by their inputs and outputs. What is the semantic of a service (`car_price_service.owls`) that takes a concept of type *Car* as input and has a concept of type *Price* as output? It might sell you a car and tell you the price afterwards, it might just as well only inform you about the price of a new car or the price of a used car. It might rent a car for the returned price. It might tell you the price of the yearly inspection

for the given car. There are many different possible interpretations. What is the semantic of a service like `car_priceauto_service.owls` that takes as input a concept of type *Car* and has outputs of type *Price* and *Auto* (which is a subclass of car)?

In our view the services of OWLS-TC2 are not described in sufficient detail to allow to perform meaningful semantic discovery on them. The problem is greatly aggravated by the fact that the services in OWLS-TC2 make use of classes in a class hierarchy but do not make use of attributes or relations. Thus, in most cases the semantic of the services is greatly underspecified and - if at all - understandable only from the informal textual documentation[11]. Overall it seems the textual descriptions of the service offers and queries are not captured well by the semantic descriptions. Query 23 for instance is informally described as "the client wants to travel from Frankfurt to Berlin, that's why it puts a request to find a map to locate a route from Frankfurt to Berlin." This request is described (`geographicalregiongeographical-region_map_service.owls`) as a request for a service with two unordered inputs of type *geographical region* and a single output of type *map*. Clearly routing services will also be found (among many others) by such a request, but we are afraid that offers and requests described at this level of detail will neither allow to demonstrate the added value of *semantic* service discovery nor to evaluate the power of matchmakers which should create this added value.

Independence of offer and request descriptions. Ideally, service offer and request descriptions should be designed independently since this is the envisioned situation in reality. Service providers describe their offers, clients query for a service with a semantic request description and a matchmaker is supposed to find the offers that match the request. We acknowledge that in laboratory settings it is sometimes desirable to artificially design the offers to match a request at various degrees. This way it can be assured that all potentially existing degrees of match occur during a test run. However, a test where the offers have been designed to match a request at hand with specific degrees runs the risk of doing nothing more than supporting the belief that a particular matchmaker *implementation* operates as expected. It does not demonstrate the power of a certain semantic description formalism or a certain matchmaking approach. Despite the fact that OWLS-TC2 claims that most services where retrieved from public IBM UDDI registries, we got the impression that for most of the queries in OWLS-TC2 the matching services have been artificially designed for that particular query. Query 4 for instance asks for the combined price of a car and a bicycle. It seems quite idiosyncratic to buy a car and a bicycle as a package, yet there are at least eleven service offers in OWLS-TC2 that precisely offer to provide the price of a package of one car and

[11] This may have been on purpose since the OWL-MX matchmaker, the matchmaker OWLS-TC was designed for, is a hybrid matchmaker that combines semantic matchmaking with traditional information retrieval techniques

one bicycle. Our impression is further backed up by the fact that the number of relevant services is quite stable for all the queries.

Conclusions. OWLS-TC2 has been developed by the effort of a single group to evaluate a particular (hybrid) matchmaker [7] and the OWLS-TC2 manual states that it has been designed to be balanced with respect to the matching filters of that matchmaker, i.e. besides performing semantic discovery it explicitly also uses classical Information Retrieval techniques. Thus OWLS-TC2 is suited to test and evaluate the features of this particular hybrid matchmaker, but for the reasons given above we do not think this test collection is suited for a broader comparative evaluation of different semantic matchmakers. Based on this finding and the discussion in Section 2 we doubt that the current setup of the S3 contest will yield meaningful results.

To put our criticism above into the correct context, we would like to acknowledge once more that sadly there is currently no better public test collection than OWLS-TC2 and that the creation of a balanced, realistic and rich, high-quality semantic service test collection involves an immense amount of effort that clearly exceeds the capabilities of any single group. The organizers of the S3 Contest have therefore stressed that such a collection can only be built by the community as a whole and that the contest and its current employment of OWLS-TC2 is only a first step in that direction. They have set up a wiki[12] to initiate a corresponding community effort. We hope that our critical analysis of OWLS-TC2 will help to motivate such community effort and will therefore ultimately help to improve the quality of the emerging collections.

3.2. Semantic Web Service Challenge

The Semantic Web Service Challenge is an initiative aiming to create a test bed for frameworks that facilitate the automation of web service mediation and discovery. It is organized as a series of workshops in which participants try to model and solve problems described in the publicly available test bed. The test bed is organized in scenarios (e.g. discovery or mediation), each one containing detailed problem descriptions. Compared to the S3 contest the number of available services (at the time of writing around a dozen) is relatively small, however the organizers put strong emphasis on providing realistic and detailed scenarios.

The Challenge organizers have realized that the lack of comprehensive evaluation and test beds for semantic web service system is one of the major blockers for industrial adoption of the used techniques. They have designed the challenge having the following ideas in mind:

- *Solution Independence.* Existing test cases often suffer from the problem that they have been reverse engineered from the solution, i.e. that the use case has been created according to the strengths of a particular solution. This hinders comparison across multiple systems. By letting the organizers not

[12] http://www-ags.dfki.uni-sb.de/swstc-wiki

directly participate and by defining rules on how new scenarios can be added the SWS Challenge tries to overcome this problem.

- *Language Neutral.* Closely connected to the above issue is the one how to describe the problem set. Using a particular formalism for describing services already implies the solution to a huge degree. In our opinion, the choice of the right level of detail to include in the service and goal descriptions in fact still constitutes one of the core research problems and should not be dictated by the test bed for an evaluation. The SWS Challenge organizers have consequently decided not to provide formal descriptions but only natural language ones.
- *No Participation Without Invocation.* Each scenario provided comes with a set of publicly available web services. On the one hand this should yield in some industrial relevance, on the other hand it provides the organizers with an unambiguous evaluation method. If a system claims to be able to solve a particular problem (e.g. discovery of the right shipment provider), this can be automatically verified by monitoring the SOAP messages exchanged.

Scenario Design. Within the research community only little consensus exists about what information should be included in a static service description and how they should be semantically encoded. The scenarios are thus described using existing technologies (WSDL, XSD, and natural language text descriptions). In the following we will explain the philisophy of the scenarios by means of the first discovery scenario[13] provided. This scenario includes five shipment services that are modeled according to the role models of order forms of existing shipment companies on the Internet. The services are backed by corresponding implementations that are part of the test bed.

The task to solve is to discover and invoke a suitable shipper for a given shipping request. The scenario contains a set of such requests which are categorized into levels of increasing difficulty. It starts with *Discovery Based on Destination* and adds weight and price criteria as well as simple composition and temporal constraints to the more difficult problems. For each request the problem description contains the expected correct solution (i.e. the list of matching services) already.

Evaluation Methodology. Solutions to a scenario are presented at the Challenge workshops. The evaluation is performed by teams composed of the workshop organizers and the peer participants. The organizers are aware, however, that this causes scalability problems if the number of participants increases and also is not strictly objective.

The evaluation approach focuses on evaluating the functional coverage, i.e. on whether a particular level of the problem could be solved by a particular approach or formalism correctly or not. The intention is to focus on the *how*, that is the concrete techniques and descriptions an approach uses to solve a problem and not

[13] http://sws-challenge.org/wiki/index.php/Scenario:_Shipment_Discovery

on the time it requires for execution, thus no runtime performance measurements are taken.

The organizers argue that in practice automatic and dynamic discovery is not widely used, thus part of the challenge is to refine the challenge and to illustrate the benefit of using semantic descriptions. The basic assumption to test is whether approaches which rely more heavily on semantic annotations will be easier adaptable to changes in the problem scenarios. Therefore the challenge does not only certify functional coverage, but initially it was planned to also assess on how *elegant* a solution can address the problems posed and how much effort was needed to proceed from a simpler to a more complex problem level.

However it turned out that it is extremely difficult to assess this in an objective manner [9]. Measurements based on counting the number of lines (or statements) of semantic description do not adequately represent the usability of an approach. Also the measurements of changes that are required to solve new problems turned out to be problematic. They worked in the beginning when all participants started with the same known set of problem levels which was then extended at consecutive workshops. However, participants entering the challenge right now have access to all problem levels right away which makes an objective assessment of the necessary change to solve the more advanced levels on top of the simpler ones impossible. It is planed to test the usability of surprise scenarios for the envisioned assessment at the next workshop.

Lessons Learned. By only describing the problems without any particular formalism in mind the SWS Challenge organizers where able to attract various different teams from different communities. Thus it successfully enables evaluation across very heterogenous solutions. By requiring a grounding in real web services a significant amount of effort was consumed both on the site of the organizers as well of the participants with problems related to standard web service technology, which are not strictly relevant when looking at discovery in isolation. This also may have discouraged potential teams more familiar with knowledge representation than with web service technology. On the other site the implementation has been proven useful to (1) disambiguate the natural language text descriptions and (2) undoubtedly show whether a participant has or has not solved a particular problem. By having an implementation, no one could change the scenario to fit their solution without failing at the automated tests based on exchanged SOAP messages.

With respect to the scenarios being described in informal natural language only, it turned out that the original scenarios were indeed ambiguous in several cases. However, during the course of usage of a particular scenarios these ambiguities where discovered by the participants and could subsequently be resolved by the authors of the particular scenario. Our experience shows that this way even scenarios described in natural language only become sufficiently well-defined over time. Usually the implementation also does disambiguate a scenario, however it is not the most efficient way to find out about the intention of a particular aspect.

3.3. DIANE Service Description Evaluation

Within the DIANE project[14], a service description language, DIANE Service Description (DSD) and an accompanying middleware supporting service discovery, composition, and invocation have been developed. DIANE is one of the projects taking part in the SWS Challenge. Besides the evaluation provided by the challenge, considerable effort has been put into devising an evaluation suite for semantic service description languages[4]. While this work is certainly not completed yet, it complements the SWS Challenge in some important aspects. The DIANE evaluation focuses on four criteria an evaluation should measure:

1. *Degree of Automation:* Are the language and the tools powerful enough to allow for automatic and correct service usage? That means: Given a service request and service offers, will the discovery mechanism find the best-matching service offer and will it be possible to automatically invoke the service based on these results?
2. *Efficiency of Matchmaking:* Is it possible to efficiently and correctly compute the matchvalue of arbitrary offers and requests?
3. *Expressiveness:* Is it possible to describe real services and real service requests in sufficient detail to meet Criteria 1? Can this be done with reasonable effort?
4. *Decoupling:* Will a discovery mechanism be able to determine similarity between service offers and requests that are developed independently of each other? In other words: If a service requester writes his request without knowledge of the existing service descriptions, does the language offer enough guidance to ensure that suitable and only suitable services will be found by the discovery mechanism?

It is quite obvious, that these criteria require contradictory properties of the description language: While, e.g., Criterion 3 requires a highly expressive language, Criterion 2 will be the easier to meet the less powerful the language is. Service description languages thus need to strike a balance between these competing requirements.

Criteria 1 and 2 can be evaluated basically by providing a proof-of-concept implementation. We will not look at them in more detail here but instead focus on the more interesting Criteria 3 and 4. To evaluate these, a benchmark has been designed. This benchmark has been used for the DSD evaluation, so far. It is, however, not language specific and can be used for other approaches, too.

To evaluate how well a service description language meets Criterion 3, a set of real world services is needed. As mentioned earlier in the paper, the example of the OWL-S TC shows that meaningful real world services are apparently not easy to come by. In particular, meaningful real world services that are described in sufficient detail are scarcely available. For our benchmark, we therefore chose a different approach: A group of test subjects not familiar with semantic web technology were asked to formulate service requests for two different application domains.

[14] http://hnsp.inf-bb.uni-jena.de/DIANE/

We have chosen a bookbuying and train ticket scenario with typical end user requests as one domain and a travel agency looking for external services that can be included in applications as the second domain. The queries the test subjects devised were formulated in natural language. This resulted in about 200 requests. In preparation of the benchmark, domain experts developed ontologies they deemed necessary to handle the two domains (books, money, trains, ...). Subsequently, the experts attempted to translate the requests into DSD and computed how many requests could be directly translated, how many could be translated but required extensions of the ontologies and how many could not be appropriately expressed using the language constructs provided by DSD. These three values measure how well the language is able to describe realistic services of different types.

To evaluate whether decoupled description of offers and requests is possible, a number of the test subjects were given an introduction to DSD. They were subsequently divided into two groups that were not allowed to communicate with each other. The groups were then asked to formulate service offers and requests, respectively, from a given natural language description. The resulting DSD description were then evaluated by the matcher and precision and recall of the matchmaking were determined. High values for both parameters indicate that it is indeed possible to decouple offer and request description. A summary of the benchmark queries and results can be found online[15].

3.4. Other Approaches

The annual IEEE Web Service Challenge[16] [3] is similar in spirit to the S3 Matchmaker Contest, but focussed rather on syntactic or low level semantic matchmaking and composition based on matching WSDL part names whereas we focus on explicit higher level semantics.

Toma et al. [15] presented a framework for the evaluation of semantic matchmaking frameworks by identifying different aspects of such frameworks that should be evaluated: query and advertising language, scalability, reasoning support, matchmaking versus brokering and mediation support. They evaluate a number of frameworks in the service as well as the grid community with regard to these criteria. The focus of the work is rather on the excellent survey than on the comparison framework itself. While the framework does provide guidance for a structured comparison, it does not offer concrete test suites, measures, benchmarks or procedures for an objective comparative evaluation.

In her PhD thesis [1], Åberg proposes a platform to evaluate service discovery in the semantic web. However, her platform is rather a software architecture to provide some guidance in the development of SWS frameworks than a real evaluation platform and it does not become clear how this platform can help to comparatively evaluate different web service frameworks. Despite of some interesting starting points she does not provide a comprehensive framework (neither in theory nor practice) that can be used for the evaluation of different discovery

[15] http://hnsp.inf-bb.uni-jena.de/DIANE/benchmark/
[16] http://www.ws-challenge.org/

	S3 Contest	**SWS-Challenge**	**DIANE**
Scope of evaluation			
Runtime performance	+	–	–
Framework tool support and usability	–	–	o
Expressivity of formalism and matchmaking	–	+	+
Supported level of decoupling	–	–	+
Quality of evaluation			
Neutral to formalism	o	+	+
Independent from solution	–	+	o
Realistic and complex use cases	–	+	o
Large test set	+	–	o

TABLE 1. Preliminary comparison of complementary strengths of the existing efforts.

approaches and furthermore ignores related approaches (like the SWS-Challenge or the S3 Matchmaker Contest) completely.

Moreover we have looked into the evaluation results of various SWS research projects (see for instance [12, 11, 2]). Many have spent a suprisingly small share of resources on evaluation. For example RW^2, an Austrian funded research project[17], has implemented different discovery engines for request and service description in different logical languages, respectively different granularity. However as evaluation only a relatively small set of a couple of dozen handcrafted services exist. The EU projects DIP and ASG have also developed similar discovery engines. With respect to evaluation they quote industrial case studies, however, in essence those are also just a small set of service descriptions. Moreover due to intellectual property rights restrictions the situation is even slightly worse, since not all descriptions are publicly available and a comparative evaluation is thus impossible.

3.5. Conclusions

Our survey has shown that - despite of the amount of attention that SWS discovery and matchmaking research receives - surprisingly little effort is devoted to experimental and comparative evaluation of the various approaches. We found only three approaches that intensively deal with SWS discovery evaluation in particular. Table 1 shows a schematic and simplified comparison of the different strengths of these approaches which is only meant to give a very high level summary of the extensive treatment above. All approaches can only be seen as starting initiatives in the right direction. The SWS-Challenge is currently the best established initiative in the field, but the S3 Matchmaker Contest and the DIANE evaluation complement it in important aspects. In particular the notions of *decoupling* the

[17] http://rw2.deri.at/

creation of offer and request descriptions, of involving *inexperienced users* in devising descriptions and all aspects related to *runtime performance* comparisons are not covered by the challenge so far.

4. Related Work

The existing approaches to evaluate SWS discovery have been covered extensively above. However, these approaches have not provided a critical review of the evaluation process itself so far. In contrast, the evaluation of evaluation in traditional information retrieval has been subject of a number of studies, e.g. by Saracevic [10] or Voorhees [17], but as we argue in Section 2, the results can not be directly applied to the domain of SWS discovery. Similar meta-evaluations have not been done in the domain of SWS discovery so far except for the work by Tsetsos et al. [16], which is the one most closely related to ours. We share the author's opinion that there is a lack of established evaluation metrics, methodologies and service test collections and agree with them that further analysis is needed to understand whether and how well-known metrics like precision and recall can be applied to service discovery. Tsetsos et al., however, focus on the weaknesses of coarse-grained binary relevance judgements and suggest to use multi-valued relevance judgements instead to exploit the fact that most service matchmakers support different degrees of match instead of a binary match/fail decisions. In contrast, we provided an in-depth discussion why the Cranfield paradigm is not applicable well to SWS discovery evaluation and presented a comprehensive survey and discussion of current service discovery evaluation efforts.

5. Directions for Future Work

5.1. Making Existing Evaluations More Transparent

We found that the existing evaluations generally lack a formal underpinning and fail to clearly discuss the intention behind their design. This makes an objective comparison difficult. As a result of our analysis in Section 3 we derive a preliminary set of questions which should be answered by any evaluation approach.

Assumptions of the test-bed or evaluation. Every evaluation should explicitly state these in order to make its results comparable:

- What are the assumptions on the formalisms / logical language used?
- What is the scope for discovery? E.g. is discovery only concerned with static descriptions or does it also involve dynamic communication?
- What is the expected outcome of the discovery? Are ranked results expected or a boolean match/nonmatch decision? If measures similar to recall or precision are used, are they defined in a meaningful way?

Dimensions measured. Evaluations should clearly indicate and motivate their scope like:

- Runtime performance, such as response time, throughput, etc.
- Scalability in terms of services.
- Complexity of descriptions required.
- Level of guarantees provided by a match. Does it assume manual post processing or support complete automation such that services can be directly executed as a result of the discovery process?
- Standard compliance and reusability. E.g. can existing ontologies be reused?

Probably because of the complexity of the matter the existing approaches only address some of the points above. We believe by providing this initial list of criteria we work towards making evaluations more transparent. This catalog might help to classify test beds and make it easier to find a suitable candidate for a planned evaluation. In addition, by answering the questions above explicitly, the designer of a test set will increase the actual value of it. This is particularly true since it helps to obtain a more objective result, given that current test sets are mainly created after a particular solution has been developed and might be biased towards that particular solution.

5.2. Towards a Standard Evaluation Methodology and Test Bed

None of the current evaluation approaches provides a comprehensive discussion of the theoretical foundation of SWS discovery evaluation even though such a discussion is necessary to justify the design decisions made by any evaluation approach and ultimately to agree upon a standard way of evaluation.

This paper provides the first comprehensive summary of the current state of the art in this field. As such it hopefully serves as an important first step towards s standard evaluation methodology and test bed for semantic service discovery. Only such an agreed-upon standard will allow to effectively compare approaches and results in an objective way, thereby promoting the advancement of the whole field as such. On the way to this standard, we identify the following rough roadmap for future work.

1. The set of possible dimensions of evaluation have to be clearly identified and motivated (*what to evaluate*).
2. For each of these dimensions suitable means of measurement have to be designed and evaluated (*which criteria to use and how to measure them*).
3. The general requirements to the evaluation process itself have to be identified (*how to achieve validity, realiability and efficiency*).
4. According to these requirements a common semantic service discovery test bed needs to be established, which ultimately allows to effectively evaluate and compare existing solutions with regard to all the dimensions in a unified way. This will clearly be a continuous effort.

6. Summary

We examined the state of the art of evaluation of SWS discovery. We discussed the general applicability of the Cranfield paradigm predominantly used for evaluation in IR and argued that this well-understood paradigm does not map directly to the domain at hand. We continued by presenting an exhaustive survey of the current evaluation approaches in SWS discovery and found that the few existing approaches use very different settings and methodologies highlighting different aspects of SWS discovery evaluation. A thorough discussion of the effects of the decisions in the design of the evaluation on the results of that evaluation is missing so far. We hope that this paper serves as a starting point towards a more systematic approach to SWS discovery evaluation and provided suggestions for future work in this direction.

References

[1] Cécile Åberg. *An Evaluation Platform for Semantic Web Technology*. PhD thesis, Department of Computer and Information Science, Linköpings Universitet Sweden, 2007.

[2] Anonymous. RW2 project deliverable D2.3: prototype implementation of the discovery component, 2006.

[3] M. Brian Blake, William Cheung, Michael C. Jaeger, and Andreas Wombacher. WSC-06: the web service challenge. In *Proceedings of the Eighth IEEE International Conference on E-Commerce Technology (CEC 2006) and Third IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2006)*, Palo Alto, California, USA, June 2006.

[4] Thomas Fischer. Entwicklung einer Evaluationsmethodik für Semantic Web Services und Anwendung auf die DIANE Service Descriptions (in German). Master's thesis, IPD, University Karlsruhe, August 2005.

[5] Donna Harman. Overview of the first Text REtrieval Conference (TREC-1). In *Proceedings of TREC-1*, Gaithersbury, Maryland, USA, November 1992.

[6] Mahboob Alam Khalid, Benedikt Fries, and Patrick Kapahnke. OWLS-TC - OWL-S service retrieval test collection version 2.1 user manual, October 2006.

[7] Matthias Klusch, Benedikt Fries, and Katia Sycara. Automated semantic web service discovery with OWLS-MX. In *Proceedings of the 5th Intern. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, Hakodate, Japan, May 2006.

[8] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[9] Charles Petrie, Tiziana Margaria, Ulrich Küster, Holger Lausen, and Michal Zaremba. SWS Challenge: status, perspectives and lessons learned so far. In *Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS2007), Special Session on Comparative Evaluation of Semantic Web Service Frameworks*, Funchal, Madeira-Portugal, June 2007.

[10] Tefko Saracevic. Evaluation of evaluation in information retrieval. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR95)*, pages 138–146, Seattle, Washington, USA, July 1995.

[11] Adina Sîrbu. DIP deliverable D4.14: discovery module prototype, June 2006.

[12] Adina Sîrbu, Ioan Toma, and Dumitru Roman. A logic based approach for service discovery with composition support. In *Proceedings of the ECOWS06 Workshop on Emerging Web Services Technology*, Zürich, Switzerland, December 2006.

[13] Walter F. Tichy. Should computer scientists experiment more? *IEEE Computer*, 31(5):32–40, May 1998.

[14] Walter F. Tichy, Paul Lukowicz, Lutz Prechelt, and Ernst A. Heinz. Experimental evaluation in computer science: a quantitative study. *Journal of Systems and Software*, 28(1), 1995.

[15] Ioan Toma, Kashif Iqbal, Dumitru Roman, Thomas Strang, Dieter Fensel, Brahmananda Sapkota, Matthew Moran, and Juan Miguel Gomez. Discovery in grid and web services environments: A survey and evaluation. *International Journal on Multiagent and Grid Systems*, 3(3), 2007.

[16] Vassileios Tsetsos, Christos Anagnostopoulos, and Stathes Hadjiefthymiades. On the evaluation of semantic web service matchmaking systems. In *Proceedings of the 4th IEEE European Conference on Web Services (ECOWS2006)*, Zürich, Switzerland, December 2006.

[17] Ellen M. Voorhees. The philosophy of information retrieval evaluation. In *Evaluation of Cross-Language Information Retrieval Systems, Second Workshop of the Cross-Language Evaluation Forum (CLEF 2001)*, pages 355–370, Darmstadt, Germany, September 2001.

Ulrich Küster
Institute for Informatics
Ernst-Abbe-Platz 2-4
D-07743 Jena
Germany
e-mail: `ukuester@informatik.uni-jena.de`

Holger Lausen
Digital Enterprise Research Institute
University of Innsbruck
6020 Innsbruck
Austria
e-mail: `holger.lausen@deri.at`

Birgitta König-Ries
Institute for Informatics
Ernst-Abbe-Platz 2-4
D-07743 Jena
Germany
e-mail: `koenig@informatik.uni-jena.de`

Whitestein Series in Software Agent Technologies, 59–75

A Framework for Dynamic Web Services Composition

Freddy Lécué, Eduardo Silva and Luís Ferreira Pires

Abstract. Dynamic composition of web services is a promising approach and at the same time a challenging research area for the dissemination of service-oriented applications. It is widely recognised that service semantics is a key element for the dynamic composition of Web services, since it allows the unambiguous descriptions of a service's capabilities and parameters. This paper introduces a framework for performing dynamic service composition by exploiting the semantic matchmaking between service parameters (i.e., outputs and inputs) to enable their interconnection and interaction. The basic assumption of the framework is that matchmaking enables finding semantic compatibilities among independently defined service descriptions. We also developed a composition algorithm that follows a semantic graph-based approach, in which a graph represents service compositions and the nodes of this graph represent semantic connections between services. Moreover, functional and non-functional properties of services are considered, to enable the computation of relevant and most suitable service compositions for some service request. The suggested end-to-end functional level service composition framework is illustrated with a realistic application scenario from the IST SPICE project.

Keywords. Semantic Web, Web Services, Service Composition, Automated Reasoning.

1. Introduction

An important benefit of the Service-Oriented Architecture (SOA) is that it enables dynamic service binding, which allows service users to discover, select and invoke services at runtime. Web services technologies [1] provide a suitable technical foundation for developing and deploying loosely coupled and reusable software components, which can be invoked through their service ports. *Web services* are distributed and programmatically accessible over standard Internet protocols, and interoperate independently of the programming languages, operating systems and hardware platforms used to implement them.

Therefore, Web services technologies offer the feature richness, flexibility and scalability needed by enterprises to profit from the SOA benefits.

Automated service discovery, selection and composition are expected to enrich the experience of service end-users through value-added services, and to allow automated processes to interact with minimal human intervention [2]. However, some work still has to be done to appropriately support dynamic and automated service discovery, selection and composition with the current Web services technologies. The automation of these tasks requires some knowledge about the services, such as: (i) description of the service capabilities, for example, in terms of the semantics of IOPEs (Input, Output, Preconditions and Effects); (ii) process model, which provides a description of the service activities, interaction protocol and exchanged messages; (iii) grounding specification of the service, which describes the coding used to map information onto messages and the protocols used to exchange these messages. These requirements are expected to be covered by defining semantic models of web services, using techniques from the *Semantic Web services* [3]. A Semantic web service is a web service described in a language with well-defined semantics. This feature of the Semantic web services enables different kinds of inference and reasoning based on the service semantic descriptions, in order to facilitate dynamic service discovery, selection and composition.

In order to tackle the challenge of service composition, most of the work done until now has focused on two main composition approaches, namely by considering functional [4, 5, 6, 7, 8] and process [9, 10, 11, 12] service aspects. The approach based on functional aspects aims at finding a sequence of atomic components described in terms of their IOPEs that matches a given query. This sequence can be executed from the start conditions provided by the query, so that the query goal is satisfied at the end of the sequence. The approach based on process aspects considers services as stateful processes with a choreography represented in terms of sequential, conditional, and iterative steps imposed by the service. These two composition approaches are complementary and form an interesting trade-off to develop solutions for service composition [13].

In this paper we focus on a framework for service composition based on functional aspects, in which services are chained according to their functional description (IOPEs). The suggested framework uses the Causal Link Matrix (CLM) formalism [14] in order to facilitate the computation of the final service composition as a semantic graph. The nodes of this semantic graph represent semantic connections between component services. By computing a CLM we increase the amount of relevant service compositions that can be obtained. The set of possible solutions are pruned, at composition time, in order to rank the service compositions according to an optimization criteria. These criteria can be defined based on the semantic similarity of component services and/or the non-functional properties of the compositions calculated by aggregating the non-functional properties of the component services.

The rest of the paper is organized as follows: Section 2 motivates our framework with application scenarios and an example; Section 3 introduces the SPICE Automatic Composition Engine (ACE) architecture in which our framework is used; Section 4 presents our framework for dynamic service composition; Section 5 comments on related work, and; Section 6 gives some final remarks.

2. Motivation

Dynamic composition of services aims at composing services that satisfy a given service request from an end-user or service developer. Services are composed of existing atomic services, which are orchestrated in the service composition.

Once dynamic service composition mechanisms are available, the service creation task performed by end-users and service developers is expected to be simplified. In this paper we specially focus on the service developer scenario that we are developing on the IST SPICE [15] project. In this scenario a service developer aims at creating a new service with some specific functional and non-functional properties. To achieve this, a formalism should be used to describe these properties in a service request, specifying these properties unambiguously to allow automatic reasoning based on the service request. After the service request is specified, the framework for dynamic service composition is capable of discovering, matching and composing a set of services that together fulfil the request. The resulting compositions are returned to the service developer, who should select the composition that best fits his needs. The service developer may adapt the selected composition further to fulfil more specific requirements. This process provides a service developer with a tool for automatically finding and composing a set of services that meet his needs, relieving him from the burden of manually dealing with the whole service creation cycle.

2.1. Example

We consider an example in which a service developer wants to develop a new service that receives a piece of text, translates it to English, and sends the translated text by SMS to a given destination number. In case no support is available for the service composition, the service developer is forced to create the service by scratch by connecting the available atomic services in the service composition implementation manually. In case an orchestration language such as WS-BPEL [16] is available, the service developer can specify the service composition in terms of an orchestration of the atomic services. In our example this corresponds to an orchestration of the translation services and SMS messaging services. The objective of our framework is to go one step further and automatically generate service compositions that cope with the service developer service request and also meet the non-functional properties (e.g., cost, response time, etc.) specified in the service request.

2.2. Service Request

Service developers specify service requests in terms of annotations that define the requested service inputs, outputs, goals, preconditions, effects and ontologies. These annotations are references to elements defined on ontologies described in OWL [17]. An example of annotated service request is:

```
<Input>
   <"LanguageOnt#Language" name="srcLang">
   <"LanguageOnt#English" name="trgtLang">
   <"LanguageOnt#Text" name="txtToTrans">
   <"TelecomOnt#PhoneNum" name="destNumber">
</Input>
<Output>
   <"TelecomOnt#AckSMS" name="AcknowledgmentSMS">
```

```
</Output>
<Preconditions/><Effects/>
<Goal>
   <"GoalOnt#translate">
   <"GoalOnt#sendSMS">
</Goal>
<Non-functional>
   <"NFPOnt#Cost" value=6>
</Non-functional>
<Ontologies>
   <"GoalOnt" "TelecomOnt" "NFPOnt" "LanguageOnt">
</Ontologies>
```

These annotations indicate that the service developer requests a service that translates a piece of text to English and sends the translated text by SMS to a given destination number. This is the running example used to illustrate our framework for dynamic service composition in this paper.

3. Automatic Service Composition Engine

The aim of SPICE is to provide a platform to support the development and deployment of innovative and value-added services during their whole life cycle. The creation and development of services is achieved in a service creation environment, which allows the manual creation of services for end-users and service developers. The service creation environment also contains an Automatic Composition Engine (ACE), which automatically constructs a service that fits a service request issued by end-users or service developers.

The SPICE ACE contains four basic components: *Semantic Analyser*, *Composition Factory*, *Property Aggregator* and *Matcher*. Figure 1 depicts the ACE architecture.

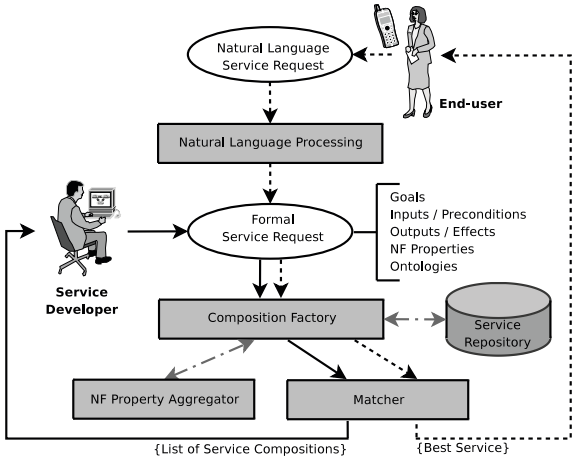


FIGURE 1. SPICE ACE Architecture

Figure 1 shows the two basic ACE usage scenarios: (i) an end-user issues a service request in natural language (at runtime) and gets the most suitable service composition,

or (ii) a service developer issues a service request in some well-defined formalism (at design-time) and gets a set of relevant service compositions.

The end-user is shielded from the complexity of the composition process by being allowed to request services in natural language. These requests are processed by the Semantic Analyser, which constructs a formal service request according to the ACE's service request formalism. The resulting formal request follows the same structure used by the service developer for defining service requests.

When a formal service request is defined, the Composition Factory queries the service repository for a service that matches the service request. If a match exists on the repository, the matching service is returned. In case no match is found, the Composition Factory creates a composite service that matches the request. In principle, the Composition Factory may generate multiple alternative compositions that match a service request.

Services and service requests are characterized by their functional and non-functional properties. Functional properties are the services' goals, inputs, outputs, preconditions and effects. These properties are used to perform the service discovery, matching and composition. Examples of non-functional properties are cost, security, performance, reliability, etc. Non-functional properties are used to limit the space of compositions that fulfil the service request, and to rank the generated set of compositions. Service and service request descriptions also contain the domain ontologies used to define the functional and non-functional properties in an unambiguous form.

The Composition Factory uses the Property Aggregator to compute the non-functional properties of service compositions each time a new service is added to a service composition. The non-functional properties of the resulting service composition are calculated by aggregating the non-functional properties of the atomic component services.

The set of generated service compositions is then passed to the Matcher component, which matches each service composition with the service request, using the aggregated non-functional properties and the measures of semantic similarity. In the scenario where the end-user requests a service, the best matching is returned to the end-user. This matching is obtained by taking the user's profile and context information into consideration, which are managed by the SPICE platform. In the scenario where the service developer issues a service request, the full set of generated compositions is returned, possibly ranked taking into account the resulting aggregated non-functional properties and/or the measures of semantic similarity.

4. Dynamic Web Service Composition

The Composition Factory component is responsible for the creation of service compositions based on a formal service request, and is the focus of this section. After receiving the developer's service request, the Composition Factory queries the service repository in order to retrieve an unordered set of services required to compute the service composition. Semantic connections between web services are stored on a CLM^+, which is then used to

compute the semantic graph-based composition that represents the possible service compositions matching the service request. Figure 2 gives an overview of the steps performed by our dynamic service composition framework.

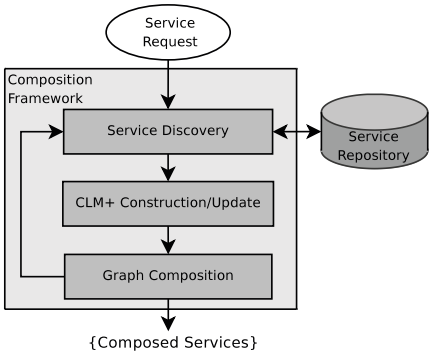


FIGURE 2. Service Composition Framework

4.1. Causal Links

When using functional composition approaches, semantic connections between different component web services are the main issue to be handled in order to create new value-added web services. These connections are mainly useful to semantically link output to input parameters of web services, creating in this way simple sequential compositions of web services. A composition is defined as an ordered set of web services in which the web services of this set are semantically linked to each other.

Input and Output parameter types of semantic web services are concepts defined in an ontology $\mathcal{T}$. These parameter types can be represented by using some standard language, such as, e.g., OWL-S [18] (at profile level), WSML [19] (at capability level), or SA-WSDL [20]. Retrieving the semantic connection between two Web services s_x and s_y is similar to discovering the semantic similarity between an output parameter Out_s_y of s_y and an input parameter In_s_x of s_x (or vice-versa). Consequently, our goal is to find a matchmaking [21] function between two knowledge representations encoded using the same ontology $\mathcal{T}$. Causal links[1] [14] between web services not only value these semantic matchmaking functions, but also measure the quality of semantic links between web services. In other words, a causal link (see figure 3) describes a semantic relation between an output parameter $Out\_s_y \in \mathcal{T}$ of a service s_y and an input parameter $In\_s_x \in \mathcal{T}$ of a service s_x. Thereby s_x and s_y are semantically and partially linked according to a matchmaking function $Sim_{\mathcal{T}}(Out\_s_y, In\_s_x)$. The matchmaking function $Sim_{\mathcal{T}}$ determines the matchmaking type [23, 24] between these two parameters, and can have the following values:

[1] In AI planning area, causal links are sometimes called *protection intervals* [22].

- **Exact (≡)** if the output parameter Out_s_y of s_y and the input parameter In_s_x of s_x are equivalent concepts; formally, $\mathcal{T} \models Out\_s_y \equiv In\_s_x$.
- **PlugIn (⊑)** if Out_s_y is sub-concept of In_s_x; formally, $\mathcal{T} \models Out\_s_y \sqsubseteq In\_s_x$.
- **Subsume (⊒)** if Out_s_y is super-concept of In_s_x; formally, $\mathcal{T} \models In\_s_x \sqsubseteq Out\_s_y$.
- **Intersection (⊓)** if the intersection of Out_s_y and In_s_x is satisfiable; formally, $\mathcal{T} \not\models Out\_s_y \sqcap In\_s_x \sqsubseteq \bot$.
- **Disjoint (⊥)** if Out_s_y and In_s_x are incompatible; formally, $\mathcal{T} \models Out\_s_y \sqcap In\_s_x \sqsubseteq \bot$.

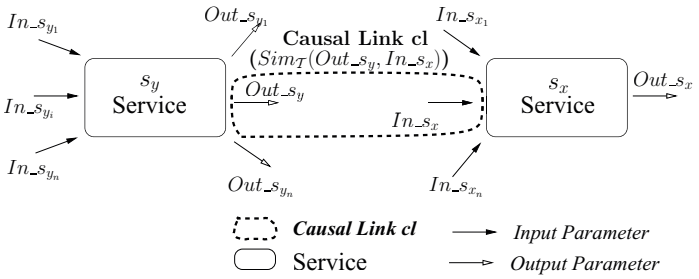


FIGURE 3. Causal Link.

Since a causal link is related to a logical dependency among input and output parameters of different web services, [14] defines a causal link as a triple $\langle s_y, Sim_{\mathcal{T}}(Out\_s_y, In\_s_x), s_x\rangle$. s_x and s_y refer to two web services in a set of available web services S_{Ws}. The concept Out_s_y is an output parameter of the service s_y whereas the concept In_s_x is an input parameter of the service s_x. The matchmaking function $Sim_{\mathcal{T}}$ returns the matching type depending on the matching degree between the concepts $Out\_s_y, In\_s_x \in \mathcal{T}$. A causal link $\langle s_y, Sim_{\mathcal{T}}(Out\_s_y, In\_s_x), s_x\rangle$ implies that (a) s_y precedes s_x, since an output of s_y is consumed by an input of s_x, and (b) no web service call is planned between s_x and s_y.

Definition 1. *(Valid Causal link)*
A causal link $\langle s_y, Sim_{\mathcal{T}}(Out\_s_y, In\_s_x), s_x\rangle$ is valid iff $Sim_{\mathcal{T}}(Out\_s_y, In\_s_x)$ is not a $Disjoint$ matchmaking.

The matchmaking type returned by the causal link is useful to value the possible semantic connection between two web services and also to compare links. Considering two web services s_y and s_z with their respective output parameters Out_s_y and Out_s_z. Considering a service s_x so that both Out_s_y and Out_s_z semantically match with In_s_x, $Sim_{\mathcal{T}}$ is able to quantify the two connections (Out_s_y, In_s_x) and (Out_s_z, In_s_x) and also to order them with respect to the matchmaking.

Although the matchmakings $Exact$, $PlugIn$, and $Disjoint$ can be used without any change to value causal links in a web service composition, causal links valued as Intersection or Subsume (also known as non-robust causal links) need some refinements to

be fully efficient for causal links composition. Further details on web service composition with non-robust causal links are given in [25].

Since a composition of web services consists of a partial order of web services in which these services are semantically chained by causal links, web service composition can be considered as a composition of causal links. Therefore in this paper we simply reuse and extend the CLM model to store the causal links that are relevant for service composition.

4.2. Service Discovery

We perform service discovery based on the service request goals in order to discover candidate services for the composition. To discover these services we assume that all the services in the service repository have a semantic goal description and references to ontologies, which can be used to search and discover the relevant services. Our framework does not support service discovery; we simply assume the availability of functionality to perform goal-based discovery in the service execution environment. In SPICE, ontology-based discovery is expected to be supported by a discovery facility.

In our running example, two main goals have been defined for the service request: $GoalOnt\#translate$ and $GoalOnt\#sendSMS$. Using these semantic annotations, the repository is queried for existing services that cope with these goals, or services that have goals semantically close to these goals. We assume that a set of services S_{Ws} is returned, and no single service fully matches the service request. Table 1 shows a possible list of discovered services S_{Ws}, with respective inputs, outputs and non-functional properties semantic types and values.

TABLE 1. Discovered Services

Service	Input	Output	NF properties
S_1	LanguageOnt#Language LanguageOnt#English LanguageOnt#Text	LanguageOnt#EnglishText	NFPOnt#Cost 1
S_2	LanguageOnt#French LanguageOnt#English LanguageOnt#Text	LanguageOnt#EnglishText	NFPOnt#Cost 4
S_3	TelecomOnt#PhoneNum LanguageOnt#Text	TelecomOnt#AckSMS	NFPOnt#Cost 1
S_4	TelecomOnt#PhoneNum LanguageOnt#Text	TelecomOnt#AckSMS	NFPOnt#Cost 3
S_5	TelecomOnt#AckSMS	TelecomOnt#SuccessProcess	NFPOnt#Cost 1

Table 1 shows that service S_1 is responsible for translating any text in any language to English, whereas S_2 translates text from French to English. These web services refer to three simple $\mathcal{FL}_0$ ontologies, namely $LanguageOnt$, $TelecomOnt$ and $NFPOnt$. The properties of these parameters are: $EnglishText \sqsubset Text$, $French \sqsubset Language$ and $Cost \sqsubset NFProperty$.

4.3. CLM and Non-functional Parameters

We extend the definition of CLM [14] below by considering not only causal links but also non-functional parameters of services. In this way, a CLM extended with non-functional

parameters, denoted as CLM$^+$ (definition 2), can be used in the automated web service composition process by classifying web services in an appropriate way, according to the causal link and the services' non-functional parameters. All causal links are pre-computed in the CLM$^+$ to facilitate web service composition. The more valid causal links can be found, the better the solution to the functional composition problem.

Definition 2. ***(CLM$^+$)***
An extended CLM (CLM$^+$), $\mathrm{M}_{p,p}$*, is defined as a* $p \times p$ *matrix of elements* $m_{i,j}$*, which are a set of triplets* $(s_y, score, \vec{q}_{s_y}) \in S_{Ws} \times \{Exact, PlugIn, Subsume, Intersection\} \times \Re^n$ *with*

$$(s_y, score, \vec{q}_{s_y}) = (s_y, Sim_{\mathcal{T}}(Out\_s_y, c_j), \vec{q}_{s_y})$$

Columns $c_{j,j\in\{1,...,p\}}$ *and rows* $r_{i,i\in\{1,...,p\}}$ *are both labelled by* $Input(S_{Ws}) \subseteq \mathcal{T}$ *i.e., the inputs parameters of services* S_{Ws}*;* $r_i \in \mathcal{T} \cap In(s_y)$ *is the label of the* i^{th} *row such that* $In(s_y)$ *is the set of input parameters of* s_y*; and* $c_j \in \mathcal{T} \cap$ *(Input(*S_{Ws}*)) is the label of the* j^{th} *column,* $Out\_s_y \in Out(s_y)$*.*

A CLM$^+$ is a matrix with entries in $\mathcal{P}(S_{Ws} \times \{Exact, PlugIn, Subsume, Intersection\} \times \Re^n)$. Each entry of the matrix refers to a set of triples $(s_y, score, \vec{q}_{s_y})$, such that the score represents the semantic similarity between an output parameter $Out\_s_y \in Out(s_y)$ of a web service s_y and an input parameter of another web service in S_{Ws}. Therefore a CLM$^+$ pre-computes the semantic similarities between all output and input parameters of a closed set of web services, i.e., a set of relevant web services for composition. According to definition 2, a CLM$^+$ contains all enabled, legal and valid links since causal links with a $Disjoint$ score are omitted in the CLM$^+$. The value of causal links $Sim_{\mathcal{T}}(Out\_s_y, c_j)$ between two parameters in a CLM$^+$ is an element of the set $\{Exact, PlugIn, Subsume, Intersection\}$. The latter set aims at value the semantic connection between an output parameter $Out\_s_y \in \mathcal{T}$ of s_y and $c_j \in$ Input(S_{Ws}) with $Exact$ being the best and $Intersection$ being the worst.

Moreover, a CLM$^+$ aims at storing non-functional properties of web services as a vector in $\Re^n$. Therefore, any service s_y referred to in the matrix contains not only semantic connections with some other services of S_{Ws}, but also its own non-functional properties $\vec{q}_{s_y} \in \Re^n$.

Example 1. *(Illustration of the CLM$^+$ indexes and labels.)*
Let $\{S_i\}_{i,i\in\{1,...,6\}}$ *be the set of web services* S_{Ws} *(table* 1*). The number of rows and columns of the CLM$^+$ is equal to 6 according to definition* 2*. Thus rows, columns of the CLM$^+$* $\mathcal{M}$ *are indexed by* $\{1, ..., 6\}$ *and labelled by the concepts Language, French, English, Text, PhoneNum and AckSMS, respectively (table* 2*).* $\mathcal{M}$ *refers to a CLM$^+$ with entries in* $\mathcal{P}(S_{Ws} \times \{Exact, PlugIn, Subsume, Intersection\} \times \Re)$*. The non-functional properties of* $S_{i,1\leq i\leq 6}$ *refer to a simple cost value in* $\Re$*.*

The CLM$^+$ construction depends on the number of output and input parameters of web services in S_{Ws}. Suppose $\#(Output(S_{Ws}))$ and $\#(Input(S_{Ws}))$ be respectively the number of output parameters of services in S_{Ws} and the number of input parameters of services in S_{Ws}. The algorithmic complexity for the causal link matrix

TABLE 2. Labels of the rows r_i and columns c_j of the 6×6 matrix $\mathcal{M}$.

i/j index	1	2	3	4	5	6
$r_{i.label}/c_{i.label}$	Language	French	English	Text	PhoneNum	AckSMS

construction is $\theta(\#(Input(S_{Ws})) \times \#(Output(S_{Ws})))$ or $\theta((Max\{\#(Input(S_{Ws})), \#(Output(S_{Ws}))\}^2)$ so square in the worst case [26]. In other words, the CLMs$^+$ construction consists of finding a semantic similarity $score$ between the output parameters of all web services $s_y \in S_{Ws}$ and the input parameters of another web service in S_{Ws}. In case $score$ is not null, the triple $(s_y, score, \vec{q}_{s_y})$ is added in the CLM$^+$ according to definition 2. For further details [26] defines the whole process of the CLM$^+$ construction.

Example 2. *(CLM$^+$ illustration)*
The entry $m_{4,4}$ (i.e., $m_{Text,Text}$) of the matrix is equal to $\{(S_1, \sqsubseteq, 1), (S_2, \sqsubseteq, 4)\}$. In S_{Ws} there is a service S_1 with an input parameters $Text$ and an output parameter $EnglishText$, which is semantically similar to $Text$. $\langle S_1, Sim_{\mathcal{T}}(EnglishText, Text), S_3\rangle$ is a valid causal link. The $EnglishText$ and $Text$ concepts match with the Plug-in match *($\sqsubseteq$ in the matrix) according to the definition of $Sim_{\mathcal{T}}$. In this way all causal links are referred in the CLM$^+$ $\mathcal{M}$ as follows ($\equiv$ refers to the $Exact$ match):*

$$\mathcal{M} = \begin{pmatrix} \emptyset & \emptyset & \emptyset & \{(S_1,\sqsubseteq,1)\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{(S_2,\sqsubseteq,4)\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{(S_1,\sqsubseteq,1),(S_2,\sqsubseteq,4)\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{(S_1,\sqsubseteq,1),(S_2,\sqsubseteq,4)\} & \emptyset & \{(S_3,\equiv,1),(S_4,\equiv,3)\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{(S_3,\equiv,1),(S_4,\equiv,3)\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

The key contribution of CLM$^+$ is a formal and semantic model to represent and manage a relevant set of services together with their non-functional properties. Web services of S_{Ws} are discovered first, to facilitate the composition process. Therefore the set of web services S_{Ws} is closed in order to limit the dimension of CLM$^+$. Such a model enables performance analysis of the proposed compositions by considering causal links and non-functional properties of services. CLM$^+$ aims at pre-chaining web services according to their semantic similarity based on their Output/Input specification. CLM$^+$ describes all possible matchings between all the web services in S_{Ws} as semantic connections. Moreover, the CLM$^+$ model is an interesting trade-off to support development activities such as services composition verification (valid causal link) or repair, by insertion and deletion of web services in the compositions.

Once web services in S_{Ws} are semantically chained according to the causal link criteria, the composition algorithm proceeds by generating the compositions graph.

4.4. Web Service Composition Process

The actual web services composition is performed using a graph-based approach, starting from the service request outputs, and possible effects, and composing backwards in the direction of the service request inputs and possible preconditions. The composition algorithm is executed after performing service discovery and CLM$^+$ construction. The CLM$^+$ contains the services that match the service request goals, and have valid causal links. The

algorithm aims at finding a set of services with exact interface matchings ($Exact$), but other semantic matchings ($PlugIn$, $Subsume$, $Intersection$), are also considered in the graph composition algorithm. This is a realistic approach, since perfect matches may not always be possible. The non-functional properties are taken into account to optimise the search for service compositions. If a graph composition branch does not comply with the requested non-functional properties, the composition on this branch is aborted. Figure 4 depicts the graph-based service composition algorithm.

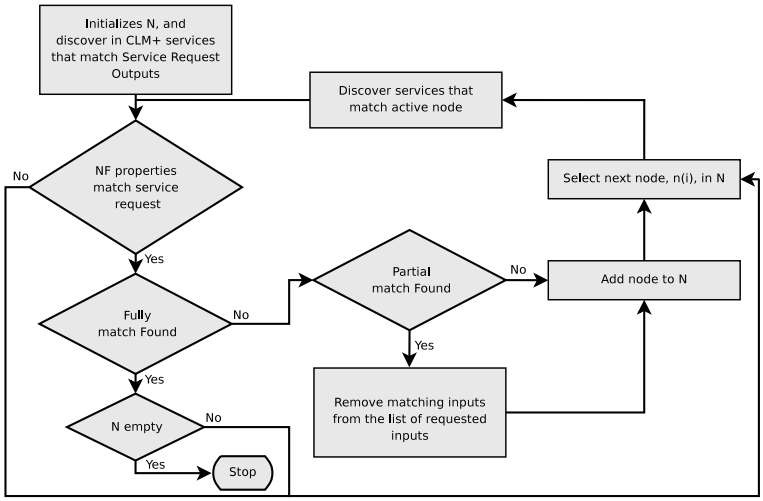


FIGURE 4. Web Service Composition Algorithm

The algorithm defines N as the set of nodes to be resolved. Each element of N represents a node with inputs that do not fully match the inputs of the service request. The algorithm initializes N with the services that provide outputs $Out(s_0)$ from the original service request. After that, the algorithm evaluates whether the retrieved set of nodes require the same inputs $In(s_0)$ as the service request. If services that match both semantic descriptions $Out(s_0)$ and $In(s_0)$ are found, and N is empty, and the services satisfy the non-functional properties of service request, the graph composition algorithm stops. In case the query returns another service S_z that does not match $In(s_0)$, but delivers $Out(s_0)$ and matches the requested non-functional properties, S_z is added to N. The algorithm then processes each node n_i of N, by searching in the CLM$^+$ for services that match the unresolved n_i inputs (and possibly preconditions). For each matching service found, the composition graph is checked, inspecting whether the composition's aggregated non-functional properties match the requested non-functional properties, if it complies the service being resolved is removed from N. If there is no match, the composition graph branch that is being resolved is pruned, meaning that the elements being resolved are removed from N, and the composition branch is removed from the graph composition. Another heuristic that can be used to avoid unrealistic compositions is to limit the graph depth, restricting in this way the maximum number of services in a service composition.

Applying the graph composition algorithm to the running example, in the first step, services that provide as output an $AckSMS$, defined on the $TelecomOnt$ ontology, are selected. Two services $\{S_3, S_4\}$ are found in the CLM$^+$ matrix. None of these services fully match the service request inputs, but they provide an $Exact$ match to one of the requested inputs ($PhoneNum$), so in these branches this requested input is set as solved for the graph composition. Given that not all the inputs have been solved, and providing that the considered non-functional property $NFPOnt\#Cost$ is satisfied, services $\{S_3, S_4\}$ are stored in N as services that provide the requested output, with an $Exact$ semantic match, but do not completely match the requested input. In the second step, S_3 is resolved by discovering services in the CLM$^+$ that provide an output semantically related to the input of S_3, namely $Text$. Services $\{S_1, S_2\}$ have been discovered to provide the text message to S_3. These services resolve the requested inputs $In(s_0)$, although only as a partial semantic match ($Plugin$), and they meet the requested non-functional property, so that the search is closed on these branches. Having reached the $In(s_0)$ on these branches, N is inspected to check whether it is empty or not. N still contains S_4 to be resolved. In the third step, S_4 is resolved, also by using services $\{S_1, S_2\}$, and the composition process is finished for this branch. In this step, the aggregated non-functional property $Cost$ of the $S_2 \rightarrow S_4$ does not meet non-functional property requirement of the service request, so this graph branch is removed from the composition graph. After this step, N is checked, and since it is empty the algorithm stops. Table 3 represents the steps discussed above, the compositions found by the graph composition algorithm, and their respective aggregated non-functional properties.

TABLE 3. Service Compositions

Step	N	Compositions	NF properties
1	$\{S_3, S_4\}$	S_3	1
		S_4	3
2	$\{S_4\}$	$S_1 \mapsto S_3$	2
		$S_2 \mapsto S_3$	5
		S_4	3
3	{-}	$S_1 \mapsto S_3$	2
		$S_2 \mapsto S_3$	5
		$S_1 \mapsto S_4$	4

Table 3 shows that the service developer obtains three alternative compositions. Further computations could be done to reduce these possibilities and determine the "best" composition as a function of the measured semantic similarity and the non-functional properties. However, we believe that the service developer should receive all found compositions that match his request, and choose the one(s) that best fit his needs himself. Nevertheless, we suggest in the sequel an algorithm to rank the generated compositions, using the compositions semantic similarity and non-functional properties values.

4.5. Ranking of Composition Results

Our web service composition algorithm aims at retrieving compositions with valid causal links and also ensuring that the non-functional properties of the service request are satisfied by the generated compositions. However, our algorithm may return more than one

composition, since some services can satisfy the same goals with different non-functional properties, or can satisfy *semantically close* goals with the same non-functional properties. In order to help service developers in their choice of service composition, we propose to rank composition results, for example, by first considering the semantic value of their causal links and after that, using the end-to-end non-functional properties of the composite services, in case the compositions have identical causal link values. To this end we assign a score for each kind of semantic connection. A causal link with an $Exact$ matching is valued to 1, a causal link with a $PlugIn$ matching is valued to $\frac{3}{4}$, a causal link with a $Subsume$ matching is valued to $\frac{1}{2}$ and a causal link with a $Intersection$ matching is valued to $\frac{1}{4}$. Such a valuation is consistent since an $Exact$ matching between an output parameter and an input parameter is more preferred than a causal link with a $PlugIn$, $SubSume$ or $Intersection$ matching.

Algorithm 1: Ranking of Composition Results.

```
1  Input: An unordered set of composition results {S_c1, ..., S_cn}.
2  Result: An ordered set of composition results (based first on causal links and second on non
           functional properties of services).
3  begin
4      foreach S_ci do
5          semantic_quality_S_ci ← Average of causal links in S_ci;
6          NF_quality_S_ci ← Function of NF properties in S_ci;
7      end
8      ({S_c1, ..., S_cn}, ≤) ← Ordering {S_c1, ..., S_cn} first by means of their semantic_quality
       and then by means of their NF_quality;
9      return ({S_c1, ..., S_cn}, ≤);
10 end
```

Non-functional properties of compositions are required in case two potential compositions of web services S_{c_i} and S_{c_j} have the same semantic quality. We overcome this issue by valuing each composition result S_{c_i} by means of a function (line 6 of algorithm 1) of the non-functional properties involved in S_{c_i}. The latter function depends on the non-functional properties of the atomic services of the composition. For instance, a sum is required to value the final cost of a composite service whereas the minimum is required to compute the throughput of a composite service. Since web services may have multiple non-functional properties, it is necessary to weight these properties, e.g., by means of user preferences. For example, an end-user may give more importance to the cost of a composite service whereas an another end-user may prefer the composite web service with the best throughput. In the service developer scenario such a ranking method could help the developer especially in case a large amount of valid composition results are returned.

5. Related Work

Recently the authors of [27] have addressed in detail the problem of interleaving web service discovery and composition, but have considered only simple workflows where web services have one input and one output parameter. In this case the web service composition

plan is restricted to a sequence of limited web services corresponding to a linear workflow of web services. The suggested solution retrieves a sequence of causal links between web services, hence a linear and total order of services. Aiming of generating a composite service plan out of existing services, in [28] a composition path is proposed that consists of a sequence of operators that compute data, and connectors that provide data transport between the operators. The search for possible operators to construct a sequence is based on the shortest path algorithm on the graph of the operators space. However, only two kinds of services (operator and connector) with one input and one output parameter are considered, which means that only the simplest case of service composition is covered. Contrary to [27] and [28], the model proposed in this paper may also consider services with more than one input and output parameter.

In [29], a composition of services is considered as a directed graph, where nodes are linked by the matching compatibility ($Exact$, $Subsume$, $PlugIn$, $Disjoint$) between input and output parameters. Based on this graph, the shortest sequence of web services from the initial requirements to the goal can be determined. This sequence corresponds to an ordered set of web services, so that this set matches all expected output parameters given the inputs provided by a user. [14] perform semantic web service composition by pre-computing the causal link matrix. Their composition strategy based on AI planning performs a regression-based approach and returns a set of correct, complete and consistent plans in which services are actions semantically linked by causal links. However, these two approaches [29, 14] compute the best composition according to the semantic similarity of output and input parameters of web services, without considering any non-functional properties of these services. A formalism and modelling tool called interface automata has been introduced in [30] to represent web services and perform compositions. Atomic services are stored as a graph where each node represents input and output parameters and edges represent web services. Each web service contains a description of its inputs, outputs, and dependencies of other web services. Web service descriptions and the graph are used to discover composition results that satisfy a service request. In case several alternative compositions are found, no optimization mechanism for selection is provided, so that in case several composition results match a request the most suitable compositions still have to be selected.

In [31] a composer is introduced to perform web services composition. The composer supports the end user to select web services for each activity in the composition and to create flow specifications to link them. Upon selecting a web service, the web services that can produce an output that could be fed as the input of the selected service are listed, after filtering based on profile descriptions. The user can manually select the service that he wants to fit in at a particular activity. After selecting all the services, the system generates a composite process in DAML-S. The composition is executed by calling each service separately, and passing the results between services according to the flow specifications. However, the composition is still semi-automatic because the user must select a web service in a restricted list. Our formal model presented in this paper aims at automating the process of web service selection according to the causal link criterion and the non-functional properties of services.

6. Final Remarks

Although web services technology is still in its infancy, some proposals are being made to enable dynamic composition of web services. Nevertheless, to the best of our knowledge, few of these proposals address both functional and non-functional properties of web services to optimize the composition process. In this paper we outlined the main challenges faced in semantic web services, i.e., dynamic composition and optimization based on non-functional properties. To this end we described a framework for the functional composition of web services. Starting from a service developer service request, we successively apply web service discovery, causal link matrix computation, web service composition and optimization based on non-functional properties of services. By computing a causal link matrix, we ensure that the obtained compositions have valid semantic connections between component web services. Finally, the set of valid service compositions is selected by considering the non-functional properties of web services involved in the composition. If a composition does not match the non-functional properties of the service request, it is neglected. Our composition approach is quite general and can be easily applied to web services described using OWL-S (service profile), WSMO (capability model) or SA-WSDL specification.

In future work, we intend to investigate how an approach based on process aspects can be combined with the approach reported in this paper. This work should allow more composition problems to be solved, increase the number of valid composition results and improve the correctness of the composition process.

Acknowledgments. This work is supported by the European IST SPICE project (IST-027617) and the Dutch Freeband A-MUSE project (BSIK 03025).

References

[1] Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web services: concepts, architectures and applications. Springer-Verlag (2004)

[2] Brodie, M.L., Bussler, C., de Bruijn, J., Fahringer, T., Fensel, D., Hepp, M., Lausen, H., Roman, D., Strang, T., Werthner, H., Zaremba, M.: Semantically enabled service-oriented architectures: a manifesto and a paradigm shift in computer science. Technical report, DERI (2005)

[3] Sycara, K.P., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. J. Web. Sem. **1**(1) (2003) 27–46

[4] Constantinescu, I., Faltings, B.: Efficient matchmaking and directory services. In: IEEE/WIC International Conference on Web Intelligence. (2003) 75–81

[5] Constantinescu, I., Faltings, B., Binder, W.: Type based service composition. In: 13th International World Wide Web Conference (Alternate track papers & posters). (2004) 268–269

[6] Klusch, M., Fries, B., Khalid, M., Sycara, K.: OWLS-MX: Hybrid OWL-S service matchmaking. In: First International Symposium on Agents and the Semantic Web. (2005)

[7] Paolucci, M., Sycara, K.P., Kawamura, T.: Delivering semantic web services. In: 12th International World Wide Web Conference (Alternate paper tracks). (2003) 829–837

[8] Sirin, E., Parsia, B., Hendler, J.A.: Filtering and selecting semantic web services with interactive composition techniques. IEEE Intelligent Systems **19**(4) (2004) 42–49

[9] Berardi, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In: ICSOC 2003. LNCS, vol. 2910, Springer (2003) 43–58

[10] Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: 12th International World Wide Web Conference, ACM Press (2003) 403–410

[11] Narayanan, S., McIlraith, S.: Simulation, verification and automated composition of web services. In: 11th International World Wide Web Conference, ACM Press (2002) 77–88

[12] Pistore, M., Roberti, P., Traverso, P.: Process-level composition of executable web services: "on-the-fly" versus "once-for-all" composition. In: ESWC 2005. LNCS, vol. 3532, Springer (2005) 62–77

[13] Bertoli, P., Hoffmann, J., Lécué, F., Pistore, M.: Integrating discovery and automated composition: from semantic requirements to executable code. In: IEEE International Conference on Web Services. (2007) 815–822

[14] Lécué, F., Léger, A.: A formal model for semantic web service composition. In: ISWC 2006. LNCS, vol. 4273 (2006) 385–398

[15] Cordier, C., Carrez, F., van Kranenburg, H., Licciardi, C., van der Meer, J., Spedalieri, A., Rouzic, J.P.L.: Addressing the challenges of beyond 3G service delivery: the SPICE platform. In: 6th International Workshop on Applications and Services in Wireless Networks. (2006)

[16] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services, version 1.1 (2003)

[17] Smith, M.K., McGuiness, D., Volz, R., Welty, C.: Web Ontology Language (OWL) guide, version 1.0. W3C. (2002)

[18] Ankolenkar, A., Paolucci, M., Srinivasan, N., Sycara, K.: OWL Web Ontology Language guide. W3C. (2004)

[19] Fensel, D., Kifer, M., de Bruijn, J., Domingue, J.: Web service modeling ontology (WSMO), W3C member submission (2005)

[20] Sivashanmugam, K., Verma, K., Sheth, A., Miller, J.: Adding semantics to web services standards. In: 1st International Conference on Web Services. (2003) 395–401

[21] Küsters, R.: Non-Standard Inferences in Description Logics. LNCS, vol. 2100. Springer (2001)

[22] Russell, S., Norvig, P.: Artificial Intelligence: a modern approach. Prentice-Hall (1995)

[23] Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: ICWS 2002. LNCS, vol. 2342, Springer (2002) 333–347

[24] Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: 12th International World Wide Web Conference, ACM Press (2003) 331–339

[25] Lécué, F., Delteil, A., Léger, A.: Applying abduction in semantic web service composition. In: IEEE International Conference on Web Services. (2007) 94–101

[26] Lécué, F., Léger, A.: Semantic web service composition through a matchmaking of domain. In: 4th IEEE European Conference on Web Services. (2006) 171–180

[27] Lassila, O., Dixit, S.: Interleaving discovery and composition for simple workfows. In: First International Semantic Web Services Symposium. (2004)

[28] Mao, Z.M., Katz, R.H., Brewer, E.A.: Fault-tolerant, scalable, wide-area internet service composition. Technical Report CSD-01-1129, University of California at Berkeley (2001)

[29] Zhang, R., Arpinar, I.B., Aleman-Meza, B.: Automatic composition of semantic web services. In: 1st International Conference on Web Services. (2003) 38–41

[30] Alfaro, L.D., Henzinger, T.A.: Interface automata. In: 8th European software engineering conference / 9th ACM SIGSOFT international symposium on Foundations of software engineering. (2001) 109–120

[31] Sirin, E., Hendler, J.A., Parsia, B.: Semi-automatic composition of web services using semantic descriptions. In: 1st Workshop on Web Services: Modeling, Architecture and Infrastructure. (2003) 17–24

Freddy Lécué
France Telecom R&D
4 Rue du clos courtel
F-35512 Cesson Sévigné
France
e-mail: freddy.lecue@orange-ftgroup.com

Eduardo Silva
Centre for Telematics and Information Technology
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
e-mail: e.m.g.silva@ewi.utwente.nl

Luís Ferreira Pires
Centre for Telematics and Information Technology
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
e-mail: l.ferreirapires@ewi.utwente.nl

Whitestein Series in Software Agent Technologies, 77–95

Composite Web Services

Kung-Kiu Lau and Cuong Tran

Abstract. Currently, composition of web services is done by orchestration. An orchestration is a workflow that combines invocations of individual operations of the web services involved. It is therefore a composition of individual operations, rather than a composition of entire web services. In this paper we propose a different approach to web service composition, whereby entire services are composed into composite services. The latter are again entire web services, that is, they can be further composed using our composition, or they can be used in an orchestration. We show how these composite services can be constructed hierarchically and used in practice.

Web service, composite service, service composition, component model

1. Introduction

In a service-oriented architecture [18], individual services are combined into a single workflow that reflects the business process in question. Although services can be defined in a general way, in practice the most widely used services are web services [13, 2].

Currently, composition of web services is carried out by orchestration [14]. An orchestration is a workflow that combines invocations of individual operations of the web services involved. It is therefore a composition of individual operations, rather than a composition of entire web services.

In this paper, we propose a different approach to web service composition, whereby entire services are composed into composite services. The latter are again entire web services, that is they can be further composed using our composition, or they can be used in an orchestration.

The key difference between our approach and web service orchestration lies in the nature of a composite web service created by our approach. A composite service has all its operations available for composition or orchestration. By contrast, in an orchestration, only the chosen individual operations of the member services are available for invocation. A composite service is a service, whereas an orchestration

is a workflow. By the same token, a composite service is also different from a choreography [14] (which is defined on a chosen set of individual operations).

Another important feature of our approach is that composition is hierarchical. This means that a composite service can be constructed step by step from sub-services in a systematic manner.

Our approach is based on our component model [11, 10]. In our model, components are built from computation units. These units provide operations but do not invoke other units, and so behave like web services. Our components are composed in a hierarchical manner by using special connectors, which we call exogenous connectors [11]. It is these connectors that make the difference between our model and other component models, and the difference between our approach to web service composition and current practice in web service composition.

2. Motivation

Currently, web service composition is done by orchestration [6]. A web service orchestration is a coordination of web service invocations, and can be represented by a workflow. It can therefore be defined as a function ORC with the following type:

$$ORC : op \times op \cdots \times op \rightarrow wf \tag{1}$$

where op is the type of operations in web services, and wf is the type of workflows for invoking a set of such operations.

An orchestration is defined using workflow languages such as BPEL [3], BPML [4] and XLANG [17]. A workflow in these languages can be converted into a web service by giving it a WSDL [13] interface. The resulting web service can then be orchestrated with other web services.

To motivate composite web services, in this section we use a simple example to show how composition is different from orchestration.

Consider a bank system with just one ATM that serves two bank consortia $BC1$ and $BC2$, each with two bank branches, $B1$ and $B2$, $B3$ and $B4$ respectively. The ATM reads the customer's card, performs a security check, identifies the customer's bank consortium and passes customer requests together with customer details to the customer's bank consortium. The customer's bank consortium checks customer details and identifies the customer's bank branch, and then passes on the customer requests and customer details to the customer's bank branch. The bank branch checks customer details and provides the usual services of withdrawal, deposit, balance check, etc.

Suppose all the elements of the bank system are available as web services, each providing appropriate operations. Then we can build a web service for the bank system by orchestrating all these web services, and converting the resulting workflow into a web service. For any particular orchestration, operations to be invoked in the web services have to be chosen, and one specific corresponding workflow is defined. Figure 1 shows one possible orchestration.

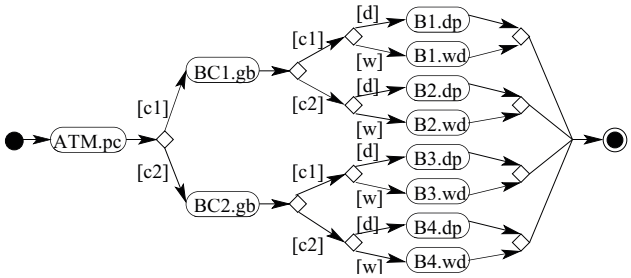


FIGURE 1. Bank orchestration.

In this workflow, the operation *pc* (processCard) of the *ATM* is invoked to identify the customer's bank consortium. The operation *gb* (getBank) of bank consortium $BC1$ or $BC2$ is invoked to get the customer's bank branch. The operations *dp* (deposit) or *wd* (withdraw) are invoked in the bank branches ($B1$, $B2$, $B3$ or $B4$). This workflow can be converted into a bank web service that provides the *deposit* and *withdrawal* operations.

Orchestration is not compositional with respect to the operations invoked. That is, given an orchestration, it is not possible to add to its set of invoked operations and hence its workflow. For example in Figure 1 it is not possible to add an invocation of *security check* to *ATM*, or a *balance check* operation to the bank branches. Any such change would require an entirely new orchestration.

This is true even if the orchestration is defined in a hierarchical workflow language like YAWL [19]. Figure 2 shows how the bank system workflow in Figure 1 can be defined in YAWL.

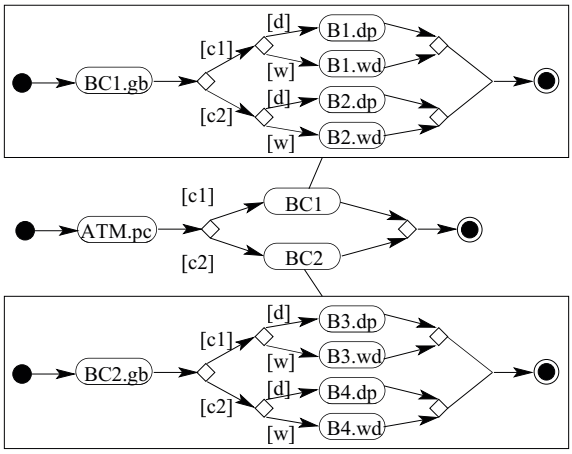


FIGURE 2. Bank with nested workflows.

To add security checks to *ATM*, it would be necessary to change the top-level workflow. To add *balance check* to bank branches, it would be necessary to change the sub-workflows for *BC1* and *BC2*.

Of course in an orchestration, it is possible to include all the operations of all the web services involved. However, such a workflow can potentially be very large, complex and cumbersome. Furthermore, it will contain many redundancies and repetitions because many sub-workflows are duplicated, as can be seen in Figures 1 and 2.

By contrast, we define a composite web service as a web service that is composed from sub-services. A composite web service is not just one orchestration, but is a web service that provides all the operations of all the sub-services, i.e. it contains all possible orchestrations of these operations. For the bank system, the composite service would have the workflow shown in Figure 3, where # denotes a parameter. This workflow is *parameterised* over all the operations of every web service involved.

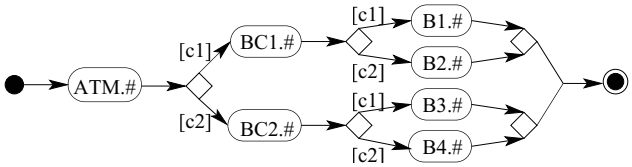


FIGURE 3. Bank composite.

3. Web Service Composition

So we want to define web service composition differently from web service orchestration. In particular, we want to define it hierarchically, that is, we want to be able to compose services into composite services, which in turn can be composed into even bigger composite services. This is illustrated by Figure 4 where web services W1 and W2 are composed into a composite service W5, and web services W3 and W4 are composed into a composite service W6. W5 and W6 are in turn composed into W7.

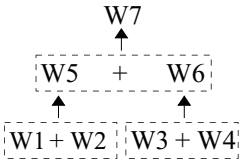


FIGURE 4. Web service composition.

A composition can be defined as a function *COMP* with the following type:

$$COMP : ws \times ws \times \cdots \times ws \rightarrow ws \tag{2}$$

where *ws* is the type of web services.

The difference between orchestration and composition can be seen clearly by comparing (1) and (2): an orchestration takes named operations (in the web services involved) as arguments and returns a workflow (for the invocations of these operations); whereas a composition takes web services and returns a (composite) web service.

Our definition of web service composition is based on a component model that we have defined [10], in particular composition in the design phase [9]. This model defines what components are, as well as composition operators for them, for different phases, namely design and deployment phases. We will show that our model can serve as a component model for web services and their composition.

3.1. A Component Model for Web Services

In our model [10], components have the distinguishing features of *encapsulation* and *compositionality*. Components are constructed from two kinds of basic entities: (i) *computation units*, and (ii) *connectors* (Figure 5). A computation unit *CU* encapsulates computation. It provides a set of methods (or operations). *Encapsulation* means that *CU*'s methods do not call methods in other computation units; rather, when invoked, all their computation occurs in *CU*. Thus *CU* could be thought of as a web service.

There are two kinds of connectors: (i) *invocation*, and (ii) *composition* (Figure 5). An invocation connector is connected to a computation unit *CU* so as to provide access to the methods of *CU*.

A composition connector encapsulates *control*. It is used to define and coordinate the control for a set of components (atomic or composite). Composition connectors can be defined for the usual control structures for sequencing and branching. A *sequencer* connector that composes components $C_1, \ldots, C_n$ can call methods in $C_1, \ldots, C_n$ in that order. A *pipe* connector is similar to a sequencer, but additionally passes the results of calls to methods in C_i to those in C_{i+1}. A *selector* connector that composes components $C_1, \ldots, C_n$ can select one component out of $C_1, \ldots, C_n$ and call methods in that component only. The control structure for looping is defined as iterators on individual composition connectors (and invocation connectors, see below). Our composition connectors are thus a Turing complete set [12, 5], for defining control flow.

Clearly composition connectors can define (and encapsulate) *workflow* for a set of connected components. They can define workflow control-flow for sequencing, branching and looping, as described in e.g. [20].

Components are defined in terms of computation units and connectors. There are two kinds of components: (i) *atomic*, and (ii) *composite* (Figure 5). An atomic

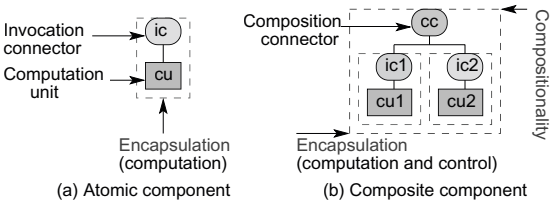


FIGURE 5. Our component model.

component consists of a computation unit with an invocation connector that provides an interface to the component. An atomic component encapsulates *computation* (Figure 5(a)). A composite component consists of a set of components (atomic or composite) composed by a composition connector. The composition connector provides an interface to the composite. A composite component encapsulates *computation* and *control* (Figure 5(b)).

An atomic component can thus be a web service, its invocation connector being the WSDL interface. A composite component can be a (composite) web service that contains sub-services as well as workflow between the sub-services. Its top-level composition connector is its interface. However, this interface cannot be described in standard WSDL since the web service now contains workflow (in the composition connector).

Our components are also *compositional*, i.e. the composition of two components C_1 and C_2 yields another component C_3. In particular, $C3$ also has the defining characteristics of encapsulation and compositionality. Thus compositionality implies that composition preserves encapsulation (Figure 5(b)).

Encapsulation and compositionality lead to *self-similarity* of composite components, as can be clearly seen in Figure 5(b). Self-similarity provides the basis for a hierarchical way of composing systems from components.

Encapsulation and compositionality result form the nature of our connectors. They are in fact *exogenous connectors* [11], and encapsulate control outside of computation units in a system. Exogenous composition connectors are defined in a hierarchical way. For example, a *sequencer* connector, or a *pipe* connector, that composes two atomic components A_1 and A_2 is clearly defined in terms of the invocation connectors in A_1 and A_2. In general, exogenous composition connectors form a hierarchy built on top of invocation connectors for atomic components. Connectors at level n for any $n > 1$ can be defined in terms of connectors at levels 1 to $(n - 1)$. Indeed, exogenous connectors have a hierarchical type system [11].

The hierarchical nature of exogenous connectors entails a strictly hierarchical way of constructing systems by composing components. In such a system, atomic components form a flat layer, and the entire control structure (of composition connectors) sits on top of this. The precise choice of connectors, the number of levels of connectors, and the connection structure, depend on the relationship between the behaviour of the individual components and the behaviour that the whole system is supposed to achieve. Whatever the control structure, however, it

is strictly hierarchical, which means that there is always only one connector at the top level. This is the connector that initiates control flow in the whole system.

As an example, the bank system can be constructed using our component model as shown in Figure 6. $P1$, $P2$ and $P3$ are pipe composition connectors;

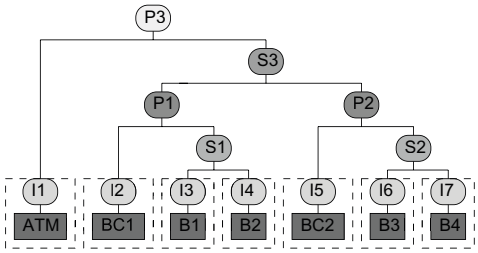


FIGURE 6. The bank system.

$S1$, $S2$ and $S3$ are selector composition connectors; and $I1 \ldots I7$ are invocation connectors. The top-level connector $P1$ is the interface to the system, and is where control flow starts.

4. Composite Web Services

Using our model as a component model for web services, we can use standard web services as atomic components, composite web services as composite components, and use the composition connectors[1] as composition operators for web services. This is illustrated in Fig 7, where two services *W1* and *W2* are composed by a composition operator *Comp* into a composite service *W3*.

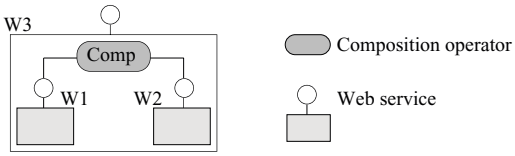


FIGURE 7. Composite web services.

W3 is a web service, just like *W1* and *W2*. However, whereas *W1* and *W2* have interfaces described in standard WSDL, *W3* has an interface that cannot be described in standard WSDL, because *W3* contains workflow embodied in the composition operator *Comp*. Therefore, in order to define *W3* as a web service, we need to extend standard WSDL in order to incorporate workflow description.

[1] In the design phase.

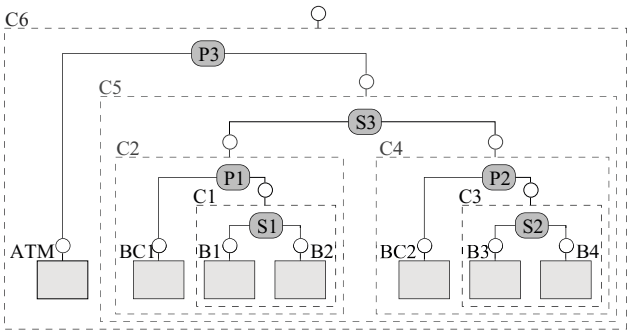


FIGURE 8. The bank composite web service.

Then we need to devise a method to generate its interface in the extended WSDL from the standard WSDL interfaces of *W1* and *W2*.

The bank system in Figure 6 can be built as a composite web service composed from standard web services for *ATM*, *BC1, BC2, B1, B2, B3* and *B4* (Figure 8). The structure of this composite is of course identical to that of the bank system in Figure 6.

The composition is hierarchical (composite services are denoted by dotted boxes): *B1* and *B2* are composed into the composite service *C1* by using the selection connector *S1*; the composite *C1* is in turn composed with *BC1* using the pipe connector *P1*, creating the composite *C2*; similarly *B3* and *B4* are composed into *C3* by using the selection connector *S2*; the composite C3 is then composed with *BC1* using the pipe connector *P2*, creating the composite *C4*; the composite *C2* is in turn composed with *C4* by using the selector connector *S3* to create the composite *C5*; the composite *C5* is composed with *ATM* by using another pipe connector *P3*, creating the composite *C6*. The composite service C6 provides all the operations offered by its sub-services.

4.1. Defining Composite Web Services

In order to define composite web services, we need to extend standard WSDL to incorporate the workflow added by connectors in composition. To this end, we define a new extensible element for WSDL documents, called *workflow*. It contains child elements which describe the details of the workflow structure. The extended WSDL document for a composite service consists of standard elements such as types, messages, portType, binding and services, together with the additional *workflow* element, as shown in Figure 9.

Under the *workflow* tag, there are extensible tags describing workflow structures. We define such a tag for each of our composition connectors. The behaviour of the connectors is defined by their implementation on the web server concerned.

The tag for each connector in turn contains child tags specifying the services (and operations) involved. If a connector provides sequential invocation, e.g. sequencer and pipe, then the child tags describe the sequence of services involved.

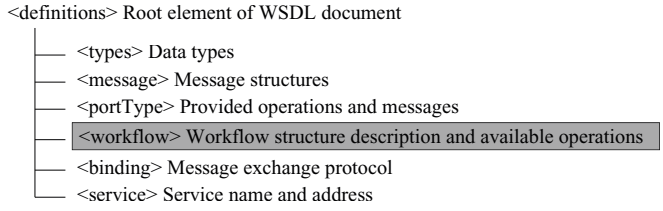


FIGURE 9. An extended WSDL document.

If a connector provides a branching structure, e.g. selector, then the child tags specify the branching condition and the corresponding services.

The schema for *workflow* consisting of pipe and selector connectors is depicted in Figure 10. This workflow element has either a pipe or choice child element. Each

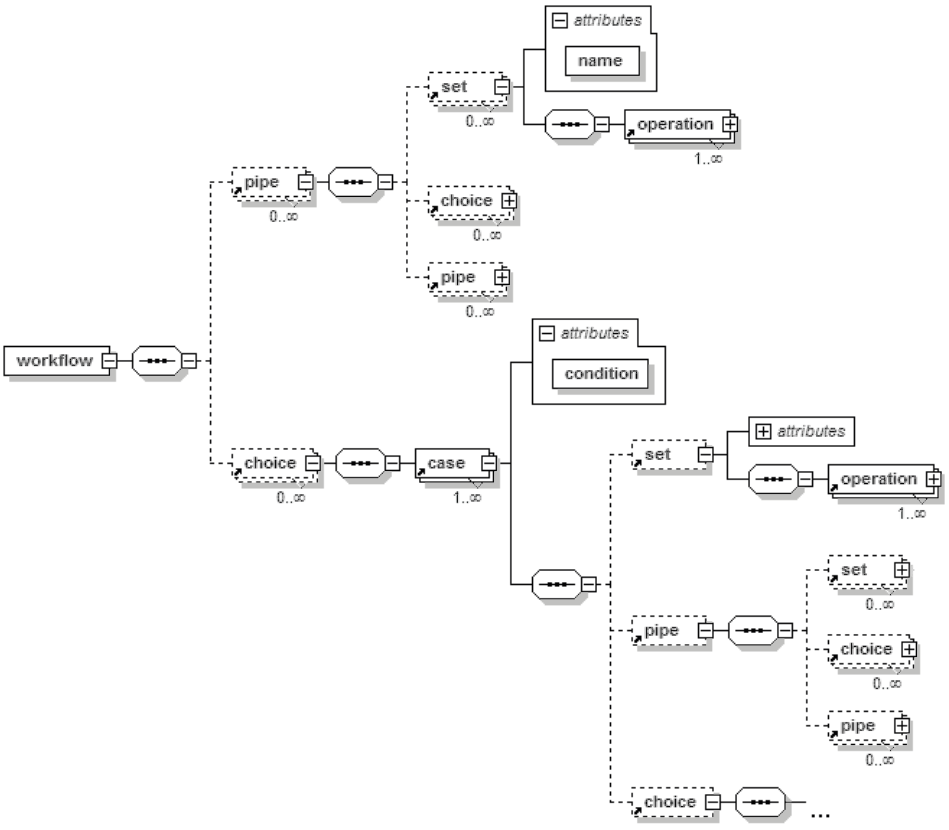


FIGURE 10. Schema for workflow and available operations.

contains a number of services (and operations). Furthermore, workflow structures (pipe and choice) may in turn contain one another.

The *pipe* tag is used to represent the pipe workflow structure provided by the pipe connector. The pipe connects a sequence of services specified by *set* tags, or other workflow structures (pipe or choice). The pipe invokes every service, or passes requests to structures, in the sequence. An invocation result is used as input to the next invocation.

The *choice* tag represents the branching workflow structure embodied by the selector connector. It contains a number of cases specified by the *case* tag. A case is a combination of a matching condition and an operation set (i.e. service) or another workflow structure. Different cases have different matching conditions. The choice workflow invokes a service or a structure if the corresponding matching condition is satisfied.

As an example, the workflow description for the composite bank service in Figure 8 can be described by the following outline:

```
<workflow>
 <pipe> <set name="ATM">
   <operation name="procCard">...</operation>
  </set>
  <choice>
   <case condition="1">
    <pipe>
     <set name="BC1">
      <operation name="getBank">...</operation> </set>
     <choice>
      <case condition="1"><set name="B1">
       <operation name="withdraw">...</operation>
       <operation name="deposit">...</operation> ...</set> </case>
      <case condition="2">
       <set name="B2">
        <operation name="withdraw">...</operation>
        <operation name="deposit">...</operation> ...</set></case>
     </choice></pipe></case>
   <case condition="2">
    <pipe>
     <set name="BC2">
      <operation name="getBank">...</operation> </set>
  ...
```

The intended meaning of this workflow is that it first invokes *any* operation of ATM, and pipes the result to the branching structure; if the result is 1, then *any* one of BC1's operations can be invoked or if the result is 2 then *any* one of BC2's operations can be invoked; the result of BC1's operation is used to compare with the branching condition; if the value is 1, then *any* one of B1's operations can be invoked, or if the value is 2, then *any* one of B2's operations can be invoked. Similarly the result of BC2's operations is used in comparison with branching condition; if the condition is 1 then *any* one of B3's operations or B4's operations will be invoked. After that, the workflow ends and the result of the last invocation is returned.

4.2. Implementing Composite Services

Given the extended WSDL document for a composite service, we need to implement the service on a web server. This implementation consists of the implementation of the intended behaviour of the workflow defined in the extended WSDL document, as well as the implementation of the interface of the composite service, also defined in the extended WSDL document.

For every composition connector, we need to implement its workflow defining its intended behaviour. To this end, we implement our connectors as Java classes which are stored as templates. Every time we use connectors to create composite services, these templates are used to generate real Java classes.

In general, for a composite service, the Java class for the top-level connector always has one operation *invoke*, that is the operation provided by the composite service to the outside world. Clients use a composite service via its *invoke* operation. Depending on the behaviour of each connector, the *invoke* operation may have different signatures. Basically, the signature of *invoke* comprises three main elements, viz. condition, operation names and operation parameters. The condition is used in a branching workflow structure for selecting sub-services. Operation names indicate which operations of the selected sub-services are invoked. Operation parameters are parameters passed to the invoked operations. Also, the signature of *invoke* includes the results returned by the composite.

The signature of *invoke* is reflected in the definitions of types, messages and portType of the extended WSDL document for a composite service. We implement message exchange style as RPC, and transport as SOAP over HTTP, in the popular manner. This information is contained inside the binding section of the extended WSDL document. The composite service address is specified at design time and contained in the service section.

The Java classes for connectors after generation are compiled and deployed to a web service engine, which is Axis [1] in our implementation. We now show our implementation for the *pipe* and *selector* connectors. For simplicity, our implementation only deals with parameters of primitive data types, e.g. string, integer, float, etc. We use String as intermediate type because other primitive types can be converted to String and vice versa.

The Pipe class template has one method:

```
invoke(String[] methods, String[] params);
```

The pipe connector receives a list of operations and a list of parameters for these operations. The *invoke* method is used to sequentially call every operation in the list. Each operation is provided by a sub-service.

If a sub-service is standard service, the connector identifies the number of parameters for every operation so that the parameters can be taken out of the parameter list and passed to the operations invoked. The connector also does type conversion for parameters if the invoked operations use primitive types different from String. If it is at the beginning or the middle of the operation list, the result of an invoked operation will be inserted into the first position of the parameter

list for subsequent operation invocations. Otherwise, the result is returned as the output of the composite. The completed operation and used parameters are thus removed from the operation and parameter lists.

If a sub-service is a composite service, the connector just passes the whole operation list at that point to the *invoke* operation of the sub-service. However, if the (composite) sub-service has a branching structure, then the connector extracts the first element of the parameter list before passing a call to the sub-service operation.

The definitions of types, messages and portType of the extended WSDL document for composite services having a pipe as the top-level connector look like the following:

```
<wsdl:types>
 <schema targetNamespace="urn:cbsd" .../>
  <complexType name="ArrayOfString">
   <sequence><element name="item" type="xsd:string"/>
   </sequence></complexType></schema>...</wsdl:types>

<wsdl:message name="invokeRequest">
 <wsdl:part name="operations" type="ArrayOfString"/>
 <wsdl:part name="params" type="ArrayOfString"/>  </wsdl:message>

<wsdl:message name="invokeResponse">
 <wsdl:part name="result" type="xsd:string"/>  </wsdl:message>

<wsdl:portType name="...">
 <wsdl:operation name="invoke"  parameterOrder="operations params">
  <wsdl:input message="invokeRequest".. />
  <wsdl:output message="invokeResponse".../> </wsdl:operation></wsdl:portType>
```

ArrayOfString is not a primitive data type, so we need to define it in the extended WSDL document. The *invoke* operation has the input message *invokeRequest* consisting of two arrays of string containing the operation and parameter lists. The output message *invokeResponse* contains the result of the *invoke* operation.

The Selector class template also has just one method:

```
invoke(String condition, String[] operations, String[] params);
```

Like the pipe connector, the selector connector receives a list of operations (provided by the sub-services) and a list of parameters for these operations. In addition, it also receives a *condition* for selecting one of the sub-services. The *invoke* method is used to call one operation in the selected sub-service, i.e. one which matches the *condition* passed to the method. If the selected sub-service is a standard service, when the selector selects whichever operation, it will identify the number of parameters and their types for the selected operation, extract parameters from the parameter list, convert to appropriate types if needed, and pass the extracted parameters to the selected operation. The result of the selected operation is the output of the composite. If the selected sub-service is a composite service, the connector will extract the first parameter from the parameter list, and put it together with the operation and parameter lists into a call to the *invoke* operation of the selected sub-service.

The definitions of types and output message of the extended WSDL document for composite services having selector as the top-level connector are similar to those for pipe connector. However, the *invoke* operation of selector has a different signature (with the addition of *condition*), which affects input message and portType definitions. These definitions are as follows:

```
...
<wsdl:message name="invokeRequest">
 <wsdl:part name="condition" type="xsd:string"/>
 <wsdl:part name="operations" type="impl:ArrayOfString"/>
 <wsdl:part name="params" type="impl:ArrayOfString"/></wsdl:message>
...
<wsdl:portType name="...">
 <wsdl:operation name="invoke" parameterOrder="condition operations params">
  <wsdl:input message="impl:invokeRequest".../>
  <wsdl:output message="impl:invokeResponse".../>
 </wsdl:operation> </wsdl:portType>
```

The binding and service sections for composite services are shown below:

```
<wsdl:binding name="..." type="...">
 <wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="invoke">  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="invokeRequest">
   <wsdlsoap:body  encodingStyle="..." namespace="urn:cbsd"
    use="encoded"/></wsdl:input> <wsdl:output name="invokeResponse">
...
<wsdl:service name="..."> <wsdl:port binding="..." name="...">
  <wsdlsoap:address  location="http://server/composite_service"/>
 </wsdl:port></wsdl:service>
```

As mentioned before, the message exchange style is RPC, and the transport is SOAP over HTTP.

4.3. Tool Support

To support our approach to service composition, we have implemented a tool. The tool can be used by a service designer to construct a composite web service, and also by a client to invoke a composite web service.

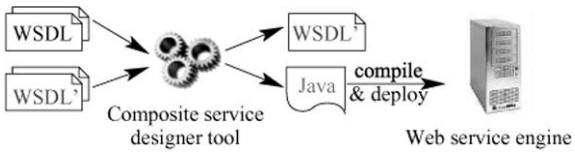


FIGURE 11. Construction process.

The process of creating composite services with our tool is illustrated in Figure 11. We start with WSDL documents of standard web services, or extended WSDL (WSDL' in Figure 11) documents for composite services as inputs. The tool generates Java classes and the associated extended WSDL document for the resulting composite service. The Java code of the composite service is compiled and the binary is deployed on to the web service engine. This construction process can

be applied hierarchically, by building (composite) services and composing them successively.

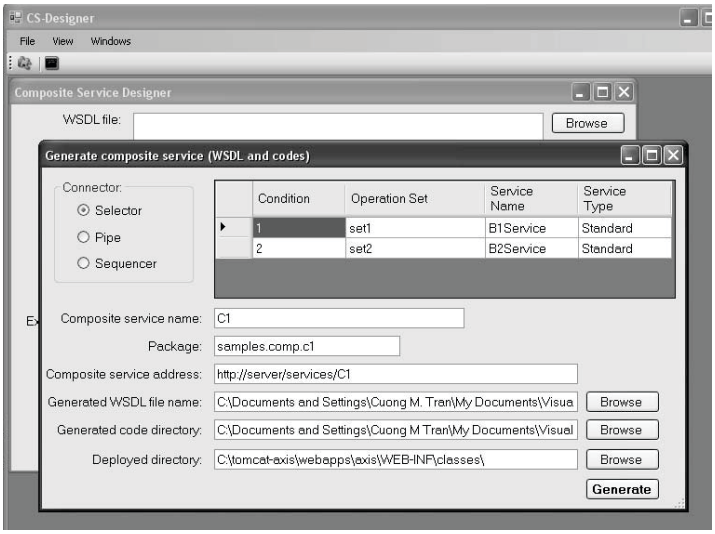


FIGURE 12. Composite service designer tool.

Figure 12 shows a screen shot of our tool being used to create a composite service. Through the user interface, the tool allows the service designer to choose WSDL or extended WSDL files as its input. The designer can also choose a desired connector, give a name to the composite service, assign composite service address, and specify the directory for the generated code.

The example in Figure 12 shows the creation of the composite web service $C1$ by composing two standard web services $B1$ and $B2$ using a selector connector. The composite service address is `http://server/services/C1`. The composite service Java code is generated and compiled. The composite service in binary is deployed to the server at the directory `c:\tomcat-axis\webapps\...\classes\`.

Figure 13 shows a screenshot of our tool being used by a client to invoke a composite service. Our tool allows clients to input a composite service description file (WSDL' file). The tool then draws a diagram of the workflow structure embodied in the composite service. The client clicks at each activity in the diagram and chooses one operation to be invoked. Based on the chosen operations, our tool generates the syntax for client calls to the composite service.

The example in Figure 13 shows a client using our composite bank service. The client can see the workflow embodied in the service, chooses the operation *withdraw* of the bank branch, and clicks the *Generate* button. The tool then shows the syntax to invoke the composite bank service, and the code for this invocation is also generated in a directory.

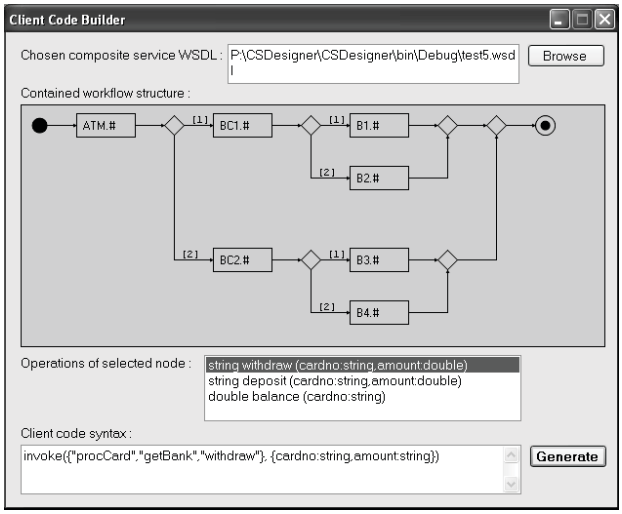


FIGURE 13. Composite service client tool.

5. Discussion

The key difference between composition and orchestration lies in the nature of a composite web service created by our approach. A composite service has all its operations available for composition or orchestration. By contrast, in an orchestration, only the chosen individual operations of the member services are available for invocation. Because it contains a workflow structure, a composite service can specify many different business logics involving the operations of its sub-services. In other words, a composite service contains many workflows, whereas an orchestration defines just one.

Our approach is distinctive compared with other current approaches. Our composite service now is truly a composite which captures entire element services and composite exists at every composition. Composite service is constructed by using our special connector as composition operators. Moreover, composite has separation between invocation control structure (given by connector). and services. This leads to our approach brings up some strong benefits. Because composition is fundamentally different from orchestration, our approach is novel. For practical purposes, we believe our approach also has some advantages over current approaches to web service orchestration.

First, our approach eases the creation of composite services. Developers need only focus on building up a structure of available services. Composition does not involve fixed operations. By allowing parameterisation of operations to be invoked, it enables clients to choose the operations based on their business logic. Thus, a composite service, once built, can be used in many different applications. In our example, the composite bank service C6 can allow multiple applications.

The second benefit of our approach is the reduction in effort of creating and maintaining web service orchestrations that belong to the same composite. Instead of incurring cost for creating and keeping separate multiple workflows working, developers of applications can just use an appropriate composite service which is already constructed to fulfil their needs. For example, in our bank composite, only the parametric workflow needs to be maintained, instead of the individual workflows that it contains. Thus our approach minimises the maintenance problem as maintenance only happens on the composite service, and the client need not change the code of his application for each business logic embodied in the composite service.

The third benefit is the hierarchical manner of building composite services. After every composition at every level of the whole system, there exist composite services. These individual composite services can be used separately by other applications. For instance, in the bank example, two composites C2 and C4 could be used in an application involving multiple bank consortia. In this case, C6 would allow customers belonging to multiple consortia to use its sub-services. Another benefit is easing service composition maintenance. Thanks to the hierarchical nature of our composition approach, if one sub-service has changed its location, only one composite service containing this sub-service is affected. The composite can be updated locally by its developer and the change can happen without requiring updates to other related services. For instance, in the bank example, if B1 changes, then only C1 is affected.

Finally, as mentioned before, our composite service is still a web service. Thus it can be used in orchestration. As shown in the previous section, our tool allows a composite service to be invoked, yielding a workflow. However, our tool is not yet integrated with standard orchestration tools. For such an integration we need to extend existing workflow designer tools such as Eclipse-BPEL. Such a tool would combine a standard WSDL processor with a processor for extended WSDL as defined in Section 4.1. Our tool can provide the processor for extended WSDL, and we are currently working on its integration with Eclipse-BPEL.

6. Related Work

Although orchestration and choreography are related to our work, we have already pointed out that our approach is fundamentally different. In orchestration, an orchestration language, such as BPEL [3], is used for defining executable workflows in XML-based format, consisting of series of activities. Every activity requires a particular service operation as input. The workflow can be deployed onto a workflow enactment system, such as the BPEL engine, which manages the workflow execution. However, existing orchestration languages like BPEL and YAWL [19] cannot describe parametric workflows as embodied in our composite web services.

Choreography focuses on describing interactions between services by specifying operations in structures such as sequence, choice, etc., using a language

like WS-CDL [8]. The approach still explicitly requires specific operations to be named in the choreography document. Furthermore, choreography of services does not result in a service which can be executed.

Aspect-oriented Web Service (AOWS) [7, 16] is web service based on AOCE (Aspect-Oriented Component Engineering). A service is enriched with an aspectual description which supports automated service discovery. This approach uses an AOConnector object which serves as a gateway to a client. The connector receives client requests and relays them to an appropriate AOWS. Their connector is unlike our composition connector because it does not define a workflow structure, and using their connector on an AOWS does not produce a service.

Web Transact [6, 15] is a framework for providing transactional features to service composition. It suggests to compose web services in hierarchical architectures. Standard web services providing similar functional capabilities are bundled using the mediator pattern to create mediator services. Mediator services are later composed to create composite services by using WSTL (Web Service Transaction Language) to specify the execution sequence of specific mediator service operations. Thus, a composite service in this approach still involves invocation of specific operations. Also, a composite does not exist at every level of composition, unlike our approach. Therefore we believe their approach is not hierarchical.

7. Conclusion

We have presented a new approach for web service composition using exogenous connectors as composition operators on web services. The composite service captures all the operations provided by the sub-services, and it allows the operations to be invoked in a defined workflow structure. A composite service thus represents a rich service, giving clients a choice of many operations. Our approach appears to have benefits compared with current approaches, especially orchestration.

In future, we plan to work on outstanding issues such as complex data structure manipulation in service communication, and error propagation among composite services.

In addition, it will be interesting to test the practicality of our approach, with regard to SOAs for larger real-world applications. To this end, we will need to investigate how to publish composite services in a suitable registry, along the lines of UDDI [13].

References

[1] Axis - web services framework web site. http://ws.apache.org/axis/.

[2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeragwarana. Business process execution language for web services - version 1.1. Technical report, IBM, 2003.

[4] A. Arkin. Business process modeling language. Technical report, BPMI Organisation, 2005.

[5] C. Böhm and G. Jacopini. Flow diagrams, Turing machines and languages with only two formation rules. Comm. ACM, 9(5):366–371, 1966.

[6] S. Dustdar and W.Schreiner. Survey of web service composition. *Int. J. Web and Grid Services*, 1(1):1–30, 2005.

[7] J. Grundy, T. Panas, S. Singh, and H. Stockle. An approach to developing web services with aspect-oriented component engineering. In *In Proceedings of the 2nd Nordic Conference on Web Services*, 2003.

[8] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web services choreography description language version 1.0. Technical report, W3C, 2004.

[9] K.-K. Lau, L. Ling, and Z. Wang. Composing components in design phase using exogenous connectors. In *In* Proc. 32nd Euromicro Conference on Software Engineering and Advanced Applications, pages 12–19, 2006.

[10] K.-K. Lau, M. Ornaghi, and Z. Wang. A software component model and its preliminary formalisation. In F. de Boer *et al.*, editor, Proc. 4th Int. Symp. on Formal Methods for Components and Objects, LNCS 4111, pages 1–21. Springer-Verlag, 2006.

[11] K.-K. Lau, P. Velasco Elizondo, and Z. Wang. Exogenous connectors for software components. In G. Heineman *et al.*, editor, Proc. 8th Int. Symp. on Component-based Software Engineering, LNCS 3489. Springer, 2005.

[12] D. Le Métayer, V.-A. Nicolas, and O. Ridoux. Exploring the software development trilogy. In IEEE Softw., volume 15, pages 75–81, 1998.

[13] E. Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI.* Addison-Wesley, 2002.

[14] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[15] P. Pires. Webtransact: A framework for specifying and coordinating reliable web services compositions. Technical report, Federal University of Rio De Janeiro, 2002.

[16] J. Hosking S. Singh, J. Grundy and J. Sun. An architecture for developing aspect-oriented web services. In *Proceedings of European Conference on Web Services, Vaxjo, Sweden*, 2005.

[17] S. Thatte. Xlang: Web services for business process design. Technical report, Microsoft, 2001.

[18] E. Thomas. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, 2005.

[19] W. van der Aalst, L. Aldred, M. Dumas, , and A. ter Hofstede. Design and implementation of the YAWL system. In 16th Int. Conf. on Advanced Information Systems Engineering, 2004.

[20] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. In Distributed and Parallel Databases, pages 5–51, 2003.

Kung-Kiu Lau
School of Computer Science, The University of Manchester
Manchester M13 9PL, United Kingdom

e-mail: {kung-kiu}@cs.man.ac.uk

Cuong Tran
School of Computer Science, The University of Manchester
Manchester M13 9PL, United Kingdom

e-mail: {ctran}@cs.man.ac.uk

Whitestein Series in Software Agent Technologies, 97–109

On the Management Requirements of Web Service Compositions

Anis Charfi, Rainer Berbner, Mira Mezini and Ralf Steinmetz

Abstract. Several works have addressed the management of individual Web Services. However, the specific management requirements of workflow-based web service compositions such as those specified in the WS-BPEL have not yet been considered. In this paper, we present several management requirements in web service compositions such as discovery and selection management, SLA and policy management, middleware services management, and management of the composite service. Supporting these requirements is crucial for providing a reliable service composition with well-defined QoS properties. We also introduce web service composition management and present our vision of having dedicated tool support for it in future WS-BPEL engines.

1. Introduction

Web services [3] that are provided by different parties can be composed to cross-organizational workflows and to value-added composite web services. Web service composition languages such as WS-BPEL (Web Services Business Process Execution Language) [4] provide a cheap means for enterprise application integration and business-to-business integration. However, we notice that whilst web service based workflows cover the functional part of the composition (control flow, data flow, etc), the management of Quality of Service (QoS for short) properties in these compositions such as performance, availability, security, and reliability have not yet been addressed appropriately. On the other hand, research has revealed that the basic web service protocol stack is not sufficient to establish web services in real-world scenarios [2] and that considering QoS requirements is crucial for a sustainable success of web services [6] including their compositions. In fact, without any guarantee regarding QoS, no enterprise will be willing to rely on external web services within critical business processes.

In this paper, we look at several management requirements in the lifecycle of web service compositions, which are mostly not supported by current composition tools. We also define web service composition management (WSCM) as the management of the composite web service, the composition-side management of the composed services, and the management of the interactions that take place within and with the composition including their QoS properties. The definition of WSCM is not specific to WS-BPEL but it works also with other composition languages. We merely assume the use of a workflow-based language to compose services that are described in terms of functionality and in terms of QoS. We will focus on WS-BPEL because it is the standard for web service composition.

It is important to note that WSCM is different from web service management because most existing works on web service management operate at the interface level, i.e., on top of WSDL [1, 21, 23]. We look at the management requirements of web service based workflows from the implementation perspective, i.e., the perspective of the user who defines and deploys such workflows. Thus, web service composition management is a form of application level management, whereby the application is implemented in a workflow language such as WS-BPEL.

The contribution of this paper is two-fold. First, it outlines management concerns that are crucial but mostly not supported in current web service composition tools. Second, it defines WSCM and explains how it differs from web service management in general. Our vision is that future orchestration engines should provide WSCM capabilities. Implementing the WSCM requirements is out of scope.

The remainder of this paper is organized as follows. Section 2 gives some background knowledge. Section 3 outlines the management requirements in web service based workflows and defines web service composition management. In Section 4, we report on related work. Section 5 concludes the paper.

2. Background

In this section, we provide some background knowledge that is relevant for understanding web service composition management.

2.1. Web Service Management and Web Service Composition

In Web Service Distributed Management (WSDM) OASIS developed a specification called Management of Web Services (MOWS) [21], which defines an additional management interface of a web service. The management interface provides information about the identity, operational state, request processing state, etc.

In [21] Web Service management is defined as an extension of enterprise application management. Following this definition, web service management has two sides: management of applications within an enterprise (internal management) and management of relationships with other web services across enterprises (external management). This distinction aims at the management of web services.

Web service composition provides a means to create a value-added web service by combining existing web services. WS-BPEL 2.0 [4] is a process-oriented web

service composition language, in which a composite web service is implemented using a workflow process. The main concepts in WS-BPEL are partners, variables, and activities. The partners are the parties that the composite web service interacts with such as clients and other web services. The variables act as containers for the data that is exchanged between the partners as well as for the process data. The activities are the units of work in the process. WS-BPEL differentiates primitive activities and structured activities. Primitive activities are atomic whereas structured activities are composite. The core of a WS-BPEL process is the set of atomic messaging activities (e.g., receive, reply, invoke), which perform interactions between the partners. Structured activities such as sequence and flow contain other activities, structuring the latter according to control flow patterns.

2.2. Service Level Agreements and Policies

Service Level Agreements (SLAs) are bilateral contracts and defined in RFC 3198 [25] as the documented result of a negotiation between a customer and a provider of a service that specifies the levels of availability, serviceability, performance, operation or other attributes of the service. An SLA contains a Service Level Specification (SLS). An SLS is a set of parameters (e.g., availability, performance, and error rate) and their values which together define the service offered to the customer. Besides the SLS, an SLA can contain pricing information, contractual information, etc. To model SLAs, we use IBM's Web Service Level Agreement (WSLA) framework [1]. WSLA is based on XML Schema and it is divided in three parts. In the section Parties, the organizations involved are described. Relevant parameters and the way how they are calculated are illustrated in the section Service Descriptions. In the section Obligations, Service Level Objectives (SLOs) are used to define criteria that have to be met by the provider.

Other QoS properties such as reliable messaging, security, and transactions are not supported by SLAs but rather by policy based languages. WS-Policy [16] is a general model and XML-based syntax that can be used to express the requirements, capabilities, and preferences of web services e.g., with respect to security (as in WS-Security [20]). A policy is a collection of assertions. There are several domain-specific assertion languages, e.g., WS-SecurityPolicy [10] defines security assertions for integrity, confidentiality, etc.

3. Web Service Composition Management

We assume that we are building a composite service using a workflow-based web service composition language by orchestrating existing web services that are described not only in terms of their functionality (as supported by WSDL) but also in terms of QoS (as supported by SLAs and policies). Further, we assume that we have some requirements on the composite web service in terms of the QoS properties that it will guarantee to its clients.

3.1. Requirements for Web Service Composition Management

We grouped several management requirements that arise when creating and deploying web service compositions into the following categories:

3.1.1. Discovery and selection management. Service composition is generally used to solve a complex problem or implement a complex business process. The complex task of the composite web service is divided into smaller tasks that can be performed by existing services.

Discovery Management is about finding appropriate web services to build the composite web service. This activity takes place primarily at design time but can also take place at runtime. Depending on the type of service to be used, discovery is done in various ways. For instance in the case of business-to-business integration the partner organizations tell each other about the services they expose. In the case of enterprise application integration the system administrator of the enterprise knows about the services that wrap a certain application. In other cases, partner web services are discovered by searching internal and external UDDI registries. A prerequisite for an appropriate discovery is a sufficient description of the services.

At first, the discovery management component of the web service composition tool has to consider the functionality (business match) and find web services that match the port type of each partner. The business match is based only on the syntax and it can be improved by using some semantic web services technology. Besides the functional match, the non-functional properties of a web service are another important criterion in web service discovery. For describing the non-functional QoS criteria of web services, SLAs and policies are the mostly used means. Such QoS descriptions allow the selection of a particular web service to be driven by QoS concerns.

When creating a new web service it is often the case that there are certain QoS requirements on that new service. For instance, the creator of the composite service may require a response time of 2 ms to be guaranteed. Inferring QoS criteria for the individual services to be composed in terms of their SLAs and policies is a complex task that requires tool support. Another example is that the creator of a composite web service may have some transaction requirements that need to be translated to requirements on the individual web services. In the case of WS-BPEL, the programmer may specify that a certain sequence with nested invoke activities has to be transactional (e.g., using a deployment descriptor [9] or policies that are attached to the process [7]. In such a case, appropriate tool support is required to restrict the selection of partners to web services with transaction support.

In other scenarios, the discovery and selection of partners may pose certain non-functional requirements on the composite service. For instance, assume that we build a service in an intra-organizational setting, which provides one operation that checks for product availability and places an order if the required product is available. If the applications wrapped up by the availability service and the order service require authentication it is then necessary to make the composite service

require authentication as well to have the authentication data that will be passed to each composed service.

In addition to the static specifications of QoS properties, the history, i.e., the runtime behavior, of a web service is sometimes necessary for the selection. For instance, if the SLA of some service specifies an availability of 80% then selection management has to gather information about each call, i.e., it has to record the runtime behavior of web services to decide which web service should be invoked. Therefore, the history of web services executions should be stored in a database. Selection based on SLAs, policies, and history can be combined.

3.1.2. Management of the composite web service. In several composition languages including WS-BPEL, the composition is exposed as a new web service, which needs to be managed. Typical management concerns in web services are lifecycle management, the startup and the shut down of the service, the number of instances, the ability of the service to provide information about itself, its identity, its current load, the number of messages it is currently processing, its dependencies on other services, error handling, forwarding of errors to some third party, etc. Since the composite web service is implemented using a workflow process (e.g., a WS-BPEL process) we need to understand what does each management concern means in terms of constructs of the workflow language, e.g., what is the relationship of service instances to process instances what is the number of messages the composite service is processing at a certain point in time.

As the lifecycle is rather implicit in WS-BPEL, when we deploy a process we start the composite service but the process might have not started yet. The service can be shutdown by undeploying the process. Moreover, the current load of the composite service can be inferred from the number of process instances.

Fault handling is another important management issue in composite web services, e.g., if an error occurs during the execution of an operation of the composite service, it is important to identify the source of that error (whether it is a process error, an error in one of the composed services, an error in the client messages, a network error, etc). Some kind of process debugger (i.e., a tool that shows the execution state of each process instance and the values of each variable) would be very helpful to identify the source of the fault so that the user can fix it and then redeploy the process.

To establish a reliable composition several QoS properties of the composite web service have to be addressed such as:

- **Availability**: Availability of a composite web service means the probability that all web services involved in the composition are available when invoked by the workflow process. A web service is considered available if it is able to respond to a request within a defined time interval.
- **Performance**: It can be measured by the throughput and the response time. Throughput means the number of requests that can be processed during a defined time slot. In the case of WS-BPEL processes throughput depends not only on the number of process instances that can run simultaneously but also

on the number of messages that one process instance can consume and on whether BPEL-specific concepts such as message correlation are used. In fact, the same process instance with correlation may be able to process multiple client requests (e.g., one for logging in, one for searching for a product, one for placing an order, etc).

The response time is the sum of transmission time and processing time and can be measured as the time for processing a request. In WS-BPEL the processing time is the period from the point where a SOAP message with a matching receive arrives at the engine to the point where a response SOAP message matching the same operation is sent using a reply activity.

One major challenge in composition management is performance modeling and performance measurement of the composite service [12]. To analyze composite services and plan the workflow control, network calculus can be used to describe the worst-case performance behavior of a composite service. By addressing capacity planning, resource usage becomes more and more important. Performance modeling and measurement are crucial to ensure that the execution of the composite service remains feasible and SLA violations due to overload are avoided.

- **Error rate**: The error rate specifies the number of processing errors within a particular time interval. The error rate of the composite web service can be calculated based on the error rates of each partner web service whilst taking into account the number of interactions with each partner. When we define the error rate for a composite service that is implemented in WS-BPEL, we have to differentiate errors that are generated by the process, errors that are thrown by partner services using fault messages, errors caused by faulty client messages, and errors due to network failures.

In addition, other QoS properties of the composite web service have to be managed such as security, reliable messaging, and transactions (cf. Section 3.1.4).

3.1.3. SLA and policy management. After the selection phase, we have a required policy (resp. SLA) for each partner (that was probably inferred from the non-functional requirements on the composite service) and a published policy (resp. SLA) for the selected service.

Moreover, the composite Web Service may have two different policies: one that is published to clients (server-side policy) and another that is used for interactions with the composed services (client-side). As the partner services may specify options in their policies, e.g., that they support either algorithm a or b for encryption, policy management should allow the process deployer to specify parameters that drive the decision on which option to choose. When such a client-side composition policy exists an effective policy [16] has to be calculated using that policy and the published policy of the partner service.

SLA and Policy management is about handling all these SLAs and policies of the partner services and the composite service. Ideally, one would like to see a

list of policies (required, published, effective) and SLAs (required, published) for each composed service and each interaction.

SLA Management should also monitor the composite service to check if the originally defined SLA is supported. It may turn out that this SLA should be modified after a certain period of time (e.g., because of the performance of a partner service that cannot be replaced, e.g., when that service is a wrapper around an internal application). Further, the SLA descriptions of the selected partner services have to be monitored from the composition side by collecting execution data for each interaction with that service to check whether the SLA was violated.

In addition, some means are needed to define how the composition runtime should behave in the case of SLA violation (e.g., notification of service provider and service consumer, sending an e-mail to an administrator, selecting an alternative service and in that case what selection strategy to follow, etc)

Based on SLAs, rankings for partner web services can be calculated for each service category (e.g., delivery web service) [6, 5]. This ranking can be later used as a foundation for the dynamic selection of a particular web service. Furthermore, IT experts can define additional rules to exclude web services that do not satisfy certain minimal QoS requirements.

3.1.4. Management of middleware concerns. There are several middleware requirements in web service compositions [9], which can be supported by WS-* based middleware services for security, transactions, reliable messaging, etc. Due to place limitations we focus only on security as a representative for the other middleware services.

Several security concerns arise in a composite web service such as the authentication of the composition in front of its partner services. Appropriate tool support is needed to specify the data (e.g., user name and password, binary keys) that should be passed to the security middleware before interacting with a partner. The security middleware will then use that data to process the SOAP message according to the WS-* specifications for security such as WS-Security and WS-Trust.

There are also confidentiality and integrity requirements for the interactions with partners that can be enforced using a WS-Security based middleware service. As there is a relationship between the security properties of an interaction and the security policies of the involved parties, the management of middleware concerns is related to policy management.

The composite web service could also have authentication requirements on its clients, i.e., it mandates incoming client requests to provide some claims; messages without appropriate user claims will be ignored. This can be the case when the client has to pay a fee for using the service. Authentication is then used to associate a contract (including a pricing model) with the client. There are further security issues such as trust, federation, secure conversations, and privacy that need to be managed and configured, e.g., if some of the partner services can be grouped into a trust domain, then the process would not have to authenticate itself separately in front of each partner (i.e., some kind of single sign-on can be introduced).

There are other middleware concerns in service compositions such as persistence, transactions, and notification, etc. For each concern, tool support is needed to define the middleware properties of each interaction (i.e., messaging activity in WS-BPEL) and also of other activities (e.g., a transactional sequence in WS-BPEL) as well as the parameters to enforce these properties.

3.1.5. Management of business aspects. Several business aspects have to be dealt with in Web Services such as enforcing legal contracts between the partners, accounting, billing, etc. Accounting is the process of tracing information systems activities to a responsible source [11] usually conducted by the service provider as a foundation for charging and billing. In the context of web service composition, there are two forms of accounting (as being the provider of the composite web service, and as being the client of the partner web services). Logging and tracing are accounting activities with the purpose of keeping track of which requests and responses have been sent to or received from clients and partner web services including the respective data.

Billing is concerned with the bills that should be given to the clients of the composite service and also the bills between the composition and the composed services. The composition may have to pay a fee for using a partner web service based on different pricing models, i.e., pay-per-use or volume-based rates. At the composition side, the management module should collect the necessary statistics about the usage of partner web services. This can be helpful to assign costs to internal business units according to the cause of the costs. Additionally, the service requester (i.e., the composition) can make use of accounting information to check the provider's invoice. Since the composition itself is a web service, which could charge clients a fee also according to various pricing models, the management module should correlate contracts and usage statistics to produce a bill.

3.2. Definition of Web Service Composition Management

We define Web Service Composition Management (WSCM) as *the management of the composite Web Service, the composition-side management of the composed services, and the management of the interactions that take place within and with the composition including their QoS properties.* In a broader sense, it includes the supporting activities that are needed to provide a reliable Web Service composition with well-defined QoS properties such as a) the discovery and selection of appropriate services to build the composition, b) the management of interactions with and within the composition in terms of SLAs, policies, middleware properties, and business aspects, and c) the management of the composite web service

It is important to note that we look at the composite Web Service from the implementation perspective, i.e., the workflow process definition is available to us and not only the interface definition of the composite web service. This perspective is different from the one taken by general web service management approaches [1, 21, 23]. The latter assume only a WSDL interface and no knowledge about the internal implementation of the web service.

Web services and web service compositions can be managed from the technical perspective and also from the business perspective [13]. From the technical perspective, web service compositions are considered as distributed computing systems. From the business perspective, they represent business processes. Thus, WSCM is positioned between traditional systems and network management on the one side, and business process management on the other side [13].

When considering WS-BPEL, the WSCM requirements mentioned so far can be supported by a WSCM module that will be hopefully part of future WS-BPEL design-time and run-time tools. The nature of the WSCM requirements and their dependency on workflow-level details make supporting them necessarily a task of a component that is part of the orchestration engine because only the engine has knowledge about the workflow constructs and their execution state. This tool should show the different partners in the composition and allow the user to define criteria for their discovery and selection as well as criteria for selecting other services if some QoS assurances are not met. The SLA and policy management view of the WSCM tool shows the SLAs and policies for each party that is involved in the composition and also the effective policy for each interaction. Further, it should provide information on the real QoS properties for each interaction via messaging activities in each process instance. The middleware concerns view shows the middleware properties of all interactions with clients and partners as well as the middleware properties of non-messaging WS-BPEL activities such as sequence and scope. The business aspects view shows contracts and also accounting and billing information. The most important view of that component is definitely the one concerned with the management of the composite web service. It includes the policies and SLAs of that service, shows values for each QoS parameter such as availability and performance, and several server-side measurements for its SLA.

4. Related Work

A lot of research has been done in the area of web service management (WSM) from the application management perspective [14, 18, 23]. OASIS proposes the *Web Services Distributed Management* specification that addresses the management of IT resources by defining web services interfaces (management through web services) [22] and the management of web services by defining messages, events state properties [21]. However, these specifications do not address the management of web services compositions at all.

In [3] Web Service Management is defined as an extension of enterprise application management, which can be seen as the task of monitoring and controlling applications in an enterprise so that they can be resilient to failures, configurable to changing needs of the business, accountable for billing and auditing, capable of performing under varying workloads, and secure to attacks [3]. Following this definition, Web Service Management has two sides: management of applications within an enterprise (internal management) and management of relationships with other

web services across enterprises (external management). The external web service management is characterized by a limited visibility and control over portions of the application. In that work the management of service compositions from the composer's view is not discussed.

The Web Service Offerings Language (WSOL) discussed in [23] supports the management of web services as well as the management of web service compositions. So this work comes close to our own. However, we believe that it is more beneficial to use widely-accepted standards, such as WS-BPEL, instead of designing new languages.

In [19], BPEL is extended with capabilities for performance measurements (e.g., logging and auditing). However, there is no complete support for the management requirements presented in our paper. Several other works such as [15] and [17] have considered QoS related non-functional properties but none of them took into consideration management issues of web service compositions.

In [24], the authors present a Web Service Management Layer (WSML), which is placed between the client application and the external web services to offer generic management functionality using aspects, e.g., billing, accounting, security and transactional support. Furthermore, the WSML proposed in [24] allows dynamically selecting and integrating web services at runtime based on rules and policies. However, there is no integration of this concept into a composition language and no focus on the management of the composition. A similar approach to the one adopted by WSML can be used together with AO4BPEL [8] to implement a WSCM layer.

The Web Service Agent Framework (WSAF) [19] achieves service selection taking into account the preferences of service consumers as well as the trustworthiness of providers. Policies are used by providers and consumers as a formal description of the offered or needed QoS. Due to possible discrepancies between the formally offered and the real QoS, service selection relying only on provider policies may lead to suboptimal service selections. To optimize service selection, the trustworthiness of provider policies has to be taken into account. Agents are used as service proxies to select services which propose the best fit between expressed offers and needs in consideration of the trustworthiness of policies. During execution agents monitor the QoS and calculate the deviation between the offered and the real QoS as a measure for the trustworthiness of the policy, which influences further service selections. This work is also not concerned with the management requirements in service composition.

5. Summary

In this paper, we illustrated several management requirements in web service compositions and defined web service composition management. Our definition is not specific to one composition language and WS-BPEL was taken as an example for illustration because it is the standard. We also explained why and how managing

a composite web service is different from the general web service management. Moreover, we argued that state of the art WS-BPEL engines are lacking support for composition management but hopefully future engines will provide support for the WSCM requirements discussed in this paper.

References

[1] Alexander Keller and Heiko Ludwig. The WSLA framework: specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1):57–81, December 2003.

[2] G. Alonso. Myths around web services. *Bulletin of the Technical Committee on Data Engineering*, 25(4):3–9, 2002.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture, and Applications*. Springer, 2003.

[4] A. Arkin, S. Askary, B. Bloch, et al. Web Services Business Process Execution Language 2.0, OASIS Standard, 11 April 2007.

[5] R. Berbner, T. Grollius, N. Repp, et al. Management of Service-oriented Architecture (SoA) based Application Systems. In *Proc. of Workshop on Enterprise Modeling and Information Systems Architectures (EMISA)*, pages 208–221, October 2005.

[6] R. Berbner, O. Heckmann, and R. Steinmetz. An Architecture for a QoS driven composition of Web Service based Workflows. In *Proc. of Networking and Electronic Commerce Research Conference*, October 2005.

[7] A. Charfi, R. Khalaf, and N. Mukhi. QoS-aware Web Service Compositions Using Non-Intrusive Policy Attachment to BPEL. In *Proc. of the 5th International Conference on Service Oriented Computing (ICSOC), Industry track, to appear*. Springer, September 2007.

[8] A. Charfi and M. Mezini. AO4BPEL: An Aspect-Oriented Extension to BPEL. *World Wide Web Journal: Recent Advances on Web Services (special issue)*, March 2007.

[9] A. Charfi, B. Schmeling, A. Heizenreder, and M. Mezini. Reliable, Secure and Transacted Web Service Composition with AO4BPEL. In *Proc. of the 4th IEEE European Conference on Web Services (ECOWS)*, pages 23–34. IEEE Computer Society, December 2006.

[10] Chris Kaler and Anthony Nadalin (Eds.). Web Services Security Policy Language (WS-SecurityPolicy) Version 1.1, July 2005.

[11] A. Committee. Accountability. `http://www.atis.org/tg2k/accountability.html`, 2001.

[12] J. Eckert, K. Pandit, N. Repp, R. Berbner, and R. Steinmetz. Worst-case performance analysis of web service workflows. In *Proc. of the 9th Conference on Information Integration and Web-based Applications Services (iiWAS)*, pages 67–77, December 2007.

[13] B. Esfandiari and V. Tosic. Requirements for web service composition management. In *Proc. of 11th HP-OVUA Workshop*, June 2004.

[14] J. Farrell and H. Kreger. Web services management approaches. *IBM Systems Journal*, 41(2):212–227, 2002.

[15] D. Gouscos, M. Kalikakis, and P. Georgiadis. An approach to modeling web service qos and provision price. In *Proc. of 4th WISE Conference*, pages 121–130, December 2003.

[16] Jeffry. Schlimmer (Eds.). Web Services Policy Framework (WS-Policy), September 2004.

[17] S. Kalepu, S. Krishnaswamy, and S. Loke. Verity: A QoS Metric for Selecting Web Services and Providers. In *Proc. of 4th WISE Conference*, pages 131–139, December 2003.

[18] V. Machiraju, A. Sahai, and A. van Moorsel. Web Services Management Network. Technical Report HPL-2002-234, HP labs, 2002.

[19] E. M. Maximilien and M. Singh. Toward autonomic web services trust and selection. In *Proc. of the 2nd International Conference on Service Oriented Computing (ICSOC)*, pages 212–221, November 2004.

[20] OASIS. Web Services Security: SOAP Message Security 1.0.

[21] OASIS. Web Services Distributed Management: Management of Web Services (WSDM-MOWS 1.0), 2004.

[22] OASIS. Web Services Distributed Management: Management Using Web Services (MUWS 1.0), 2004.

[23] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma. Management Applications of the Web Service Offerings Language (WSOL). In *15th International Conference on Advanced Information Systems Engineering - CAiSE*, pages 468–484, June 2003.

[24] B. Verheecke, M. A. Cibran, and V. Jonckers. AOP for Dynamic Configuration and Management of Web Services. In *Proc. of the International Conference on Web Services (ICWS-Europe)*, volume 2853 of *LNCS*, pages 137–151. Springer, 2003.

[25] A. Westerinen, J. Schnizlein, J. Strassner, et al. Terminology for Policy-Based Management, RFC 3198, 2001.

Anis Charfi
SAP Research CEC Darmstadt
Darmstadt, Germany

Rainer Berbner
Multimedia Communication Lab
Darmstadt University of Technology, Germany

Mira Mezini
Software Technology Group
Darmstadt University of Technology, Germany

Ralf Steinmetz
Multimedia Communication Lab
Darmstadt University of Technology, Germany

Whitestein Series in Software Agent Technologies, 111–128

BPEL^{DT} — Data-Aware Extension for Data-Intensive Service Applications

Dirk Habich, Sebastian Richly, Steffen Preissler, Mike Grasselt, Wolfgang Lehner and Albert Maier

Abstract. Aside from business processes, the service-oriented approach —currently realized with Web services and BPEL—should be utilizable for data-intensive applications as well. Fundamentally, data-intensive applications are characterized by (i) a sequence of functional operations processing large amounts of data and (ii) the delivery and transformation of huge data sets between those functional activities. However, for the efficient handling of massive data sets, a significant amount of data infrastructure is required and the predefined 'by value' data semantic within the invocation of Web services and BPEL is not well suited for this context. To tackle this problem on the BPEL level, we developed a seamless extension to BPEL—the 'BPEL data transitions.'

1. Introduction

Web services and the Business Process Execution Language for Web Services (BPEL4WS, BPEL for short) [20] are of interest to both software vendors and researchers. In this paradigm, the functionality provided by business applications is enclosed within Web service software components. Those Web services can be invoked by application programs or by other Web services via internet without explicity binding them. On top of that, BPEL has been established as the de-facto standard for implementing business processes based on Web services.

Fundamentally, a process consists of a series of activities. Therefore, BPEL offers a standardized way to describe the functional composition of services to create comprehensive process definitions. A typical service-oriented process example is the *booking a trip* application. Aside from such traditional business processes, the service-oriented approach should also be utilizable in application scenarios with special properties, since such applications can benefit from the service-oriented idea as well.

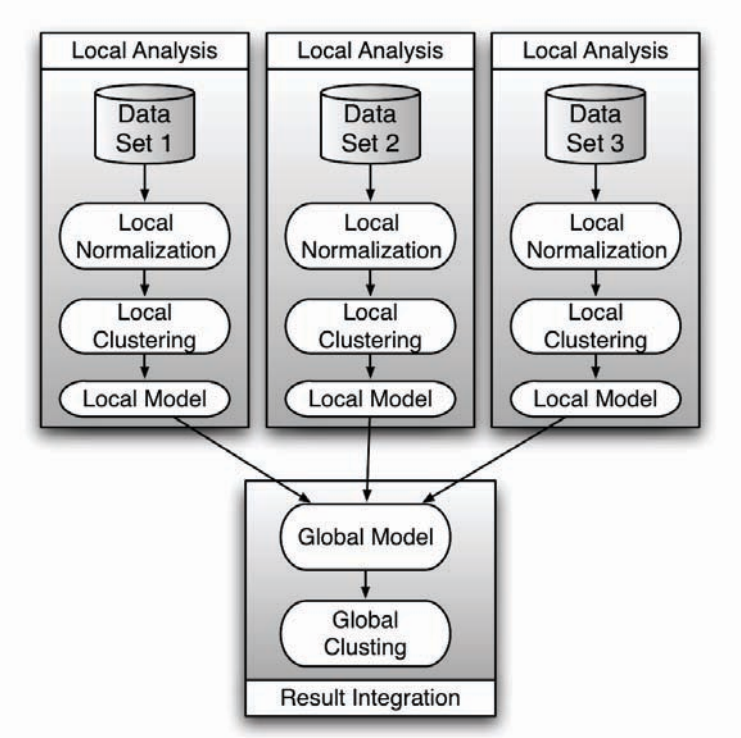


FIGURE 1. Two-Phase Clustering Approach for Gene Expression Data Sets

1.1. Data-Intensive Service Applications in SOA Environments

In the context of genome research, the method of gene expression has been used for several years. Related microarray experiments are conducted all over the world, and consequently, a vast amount of microarray data sets are produced, e.g. in biological and medical research addressing a wide range of problems.

After the conduction of the pure experiments, a data analysis process follows to extract useful knowledge. To increase the statistical significance of the extracted knowledge, researchers would like to incorporate various microarray experiments in their analysis processes. In [9], we proposed a *two-phase clustering strategy* for gene expression data sets considering this fact in a special way. A simplified view of our developed concept is illustrated in Figure 1. This analysis process offers some advantages concerning the result quality. A further important aspect is that a lot of work can be done in parallel, thus allowing the efficient process execution.

In general, the utilization of the service-oriented approach as execution environment for such processes provides some advantages with regard to process

orchestration and distributed computing. The realization of the presented analysis process would include Web services acting (i) as data providers for microarray data sets and (ii) as functionality providers for normalization and clustering strategies. The microarray data sets are usually n x m matrices, where n is the number of genes and m is the number of samples. Typical values for n and m are: $n > 20,000$ and $m > 100$. A special property of this process is that such large data sets have to be exchanged between participating Web services in the process. A further property is that it cannot be taken for granted that a participating Web service can manage the received data in the main memory during the whole processing time. Therefore, database systems are used for the internal processing of such large amounts of data.

1.2. Contribution

The current standards of Web services and BPEL are not efficiently applicable in this special context because both expose some weaknesses concerning the data aspects. One drawback is the predefined 'by value' handling of data within the service invocation of Web services. In service-oriented environments, the Simple Object Access Protocol (SOAP) is commonly used for communication with and between Web services. Embedding massive structured data sets in SOAP is possible but not a suitable solution from the performance perspective, in particular with regard to memory and scalability issues [6, 8, 18, 22]. To overcome this problem in data-centric environments, we developed the concept of *Data-Grey-Box Web Services*[8] which allow the transparent integration of specialized data propagation tools in the service invocation procedure.

The second drawback evolves from the implicitly defined data flows in BPEL and the 'by value' handling. Through the variables concept within BPEL, the concept of centralized data flows is pursued. That means the BPEL server is directly involved as a broker in the exchange of massive data sets between two participating Web services in a process. This may lead to some scalability problems on the BPEL server. To tackle this issue on the BPEL level, we propose a seamless extension to BPEL—the 'BPEL data transitions (BPEL^{DT})'—in this paper. With the help of these data transitions, the data flows are explicitly specified within BPEL processes. More detailed, our proposed BPEL data transitions represent an orthogonal data flow concept to the control flow. Furthermore, these data transitions are optimally exploited during process execution, especially in combination with our *Data-Grey-Box Web services*. Therefore, *BPEL^{DT}* and *Data-Grey-Box Web Services* from a solid foundation to support data-intensive service applications.

The remainder of the paper is structured as follows: In the next section, we go through related work, including an introduction to our developed concept of *Data-Grey-Box Web Services*. In Section 3, we describe our BPEL data transitions from a modeling and execution perspective. Implementation details and an evaluation are provided in Section 4 and Section 5. The paper closes with a summary.

2. Related Work

In this section we give a structured overview of related work. Relevant works come from the fields (i) Web services, SOAP and BPEL, (ii) Data-Grey-Box Web services and (iii) specialized data propagation tools.

2.1. Web Services, SOAP, and BPEL

Web services are an innovative architecture paradigm for applications in a service-oriented architecture (SOA). Fundamentally, Web services are considered as black-box components, since they do not offer any information on how they work; they only expose information on the structure of parameters and data they expect as input and return as result.

The Simple Object Access Protocol (SOAP) is commonly used for communication with and between Web services. The SOAP protocol defines an XML-based format for messages to be used in a Web service invocation. Such messages include a reference to the target service to invoke as well as any number of parameters and data to be transmitted to the service ('by value' semantics). Recently, the performance of the SOAP protocol has received a lot of attention. Proposed techniques try to reduce network bandwidth through compression [4, 7] or other approaches like [24, 18]. Furthermore, serialisation and de-serialisation of XML messages have been in the focus of optimization approaches [1, 2]. Furthermore, SOAP messages with attachements are an option for binary data in the form of image files or encapsulations of other XML documents [10].

On top of Web services, the Business Process Execution Language for Web Services (BPEL4WS, or BPEL for short) provides a comprehensive syntax for describing workflow logic. The BPEL language offers a number of predefined activities to express control flow patterns. The ever necessary data flow is defined implicitly by specifying variables that basically represent input and output messages of activities. In this case, the *assign* activity is of special interest because assignments are used within BPEL to manipulate variables.

To execute BPEL processes, a corresponding execution engine is required. Such a BPEL server controls the service invocations and coordinates the message exchange between Web services. Therefore, BPEL follows the concept of a centralized control flow and a centralized data flow. In this case, the BPEL server is the broker for all SOAP message exchanges between participating Web services in a process. To improve the capabilities of BPEL, a variety of extension have been proposed. One well-known extension is *BPELJ* [21] allowing to include JAVA snippets (code) in BPEL definitions. Furthemore, Maier et al. [14] proposed a similar extension to BPEL, *BPEL4SQL* supporting SQL snippets as BPEL activities and BPEL conditions. This approach can be seen as an embedded SQL approach for BPEL.

Aside from such extensions, there exist methods for (i) data-flow distribution (DFDP-WS) [19] and (ii) decentralizing the execution of composite Web services [5, 16, 17]. The DFPD-WS approach [19] extends the Web service stack with

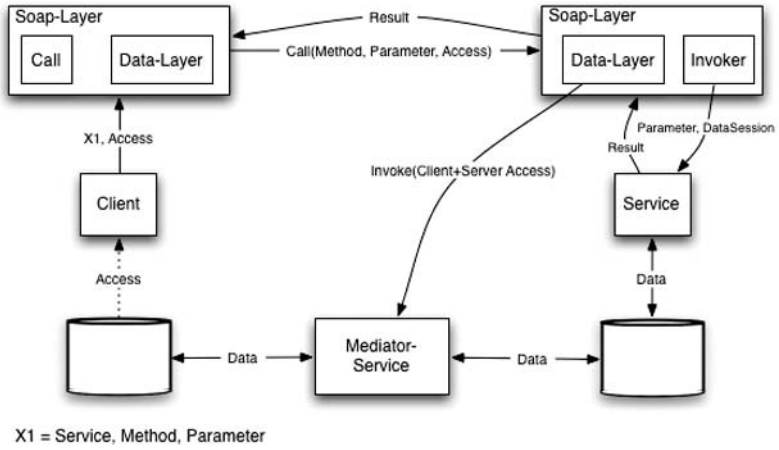


FIGURE 2. Invocation Process of Data-Grey-Box Web Services

a *Data-Flow Distribution Protocol for Web Services (DFDP-WS)* to exchange data directly without the requirement of a central broker. Nanda et al. [16] propose an approach for partitioning centralized BPEL descriptions into smaller parts that are executed by distributed BPEL engines.

2.2. Data-Grey-Box Web Services

In [8], we introduced the concept of *Data-Grey-Box Web Services*. In contrast to the original black-box Web services, we enhanced the Web service interface with an explicit data aspect offering more information about the data semantics. Aside from the separation of parameters and data in the interface description, we introduced a novel binding format for structured data. Through this new data binding, the Web service signals that data has not been transfered via SOAP but that there is a separate data layer instead. As before, regular parameters are handed over via SOAP when calling the Web service.

To handle our newly introduced data binding, we extended the SOAP framework with the integration of a novel *data layer* component, as illustrated in Figure 2 that shows the whole invocation procedure. On the client side, enhanced Web service call semantics are necessary. Aside from the transmission of the endpoint and regular parameters in the SOAP message, the client has to deliver access information as references for (i) where the input data is available (input reference) and (ii) where the output data should be stored (output reference). That means our new data binding is translated into no more than two additional parameters for access information for input and output data on the client side. These new parameters are included in the SOAP message for the invocation of Web services.

That means instead of propagating the pure data in an XML-marshaled SOAP message, we only deliver access information as data pointers in SOAP.

On the service side, our extended SOAP framework receives the SOAP message and conducts a separation into the functional aspect and the data aspect. As illustrated in Figure 2, the associated data layer calls an appropriate mediator for the data propagation based on the access information of the client and the service. While the data access information of the client can be found in the received SOAP message, the data access information for the service instance must be queried from our extended service infrastructure [8]. Fundamentally, a mediator is a neutral partner which is responsible for the data propagation between client and Web service. Examples of mediators are ETL, e.g. IBM DataStage, or data replication tools (see Section 2.3). Those mediators have to be accessible over a standard Web service interface. Advantages of the proposed mediator concept are (i) the optimized data propagation through specialized tools, (ii) the availability of an independent concept enabling an exchange of the mediators, and (iii) the release of data propagation to a third party.

If a Data-Grey-Box Web service receives and returns a data set, two data propagation tasks will be initiated. The first propagation task for the input data is conducted before the pure functionality of the service is invoked. The correlation of this input data to the Web service instance is done by our extended service infrastructure. If the input data propagation task is finished, the functionality is automatically invoked. The last step is the initiation of the data propagation task to deliver the output data to the client.

2.3. Specialized Data Propagation Tools

Fundamentally, the database research community has paid a lot of attention to the field of data exchange between different database systems. A well-known method is the ETL (Extract-Transform-Load) approach, where data from different data sources are loaded into a common data warehouse [23]. Such ETL processes consist of three parts: (i) extraction of data from the different source systems, (ii) application of a series of rules and functions to the extracted data to derive the data to be loaded, and (iii) loading of the data into a data warehouse system. This ETL approach is a data-specialized technique to efficiently transmit structured data to various different data management systems, e.g. relational or XML database systems. A further popular data propagation method is replication [13]. In database systems, this is used to provide redundancy or to balance the load across multiple database servers.

It is already possible today to include such tools in the service-oriented environment. With IBM's information server [12], for example, pre-defined ETL jobs can be provided as Web services. These Web services can then be included in the process orchestration. Disadvantages of this approach lie in the fact that (i) such data operations are explicitly integrated in the control flow and (ii) whenever such an operation is to realize the data exchange between two Web services (source and target WS) in a process, these 3 Web services are then no longer loosely coupled

but can only be used together. The reason is that the source and target WS have to be designed appropriately, i.e. the source WS do not deliver and the target WS do not receive the data via the service interface.

3. BPEL Data Transitions

Our proposed Data-Grey-Box Web Services (DGB Services) [8] are only one step in the right direction towards the efficient support of data-intensive service applications. The subsequent step is the orchestration of DGB Services to create comprehensive processes. Therefore, we present our novel BPEL data transitions (BPEL^{DT}) as a data-aware extension of BPEL in this section. These data transitions are a new explicit link type connecting several DGB Services on the data level. Fundamentally, our newly introduced BPEL data transitions are an orthogonal data flow concept to the control flow, similar to the data aspect in BPMN [3].

3.1. Modeling Perspective

From the modeling perspective, the original BPEL approach follows a two-level programming model [14, 20]. The first level (*lower level*) consists of Web services as executable software components realizing the basic activities. The upper level is often called 'programming in the large;' there, the order of the activities is orchestrated. With our explicit BPEL data transitions, we extend this programming model to a three-level approach. The three levels are:

1. *Lower Level:* This level includes Web services as executable software components; in our case, as *Data-Grey-Box Web Services* with an explicitly published data aspect.
2. *Functional Flow Modeling Level:* On this level, a domain expert models the pure functional process logic without considering data flows. The main advantage is that the domain expert can focus on the functional logic, and the data flow is modeled by data placeholders. The result on this level is a comprehensive functional process description. Such descriptions are essentially not executable.
3. *Data Flow Modeling Level:* This third level represents our extensions, where a data management expert takes the functional process description and annotates this process with all necessary data flows using our BPEL data transitions. The functional description of the process is not changed by this data flow annotation concept.

In Figure 3, a simple process with an explicit data transition between two services is depicted. In this case, the modeling of the illustrated data flow is done with the specialized data propagation tool ETL. As illustrated in the figure, the output of $WS1$ consists of two data sets (*outputSchema1*, *outputSchema2*). Then, two different transformation operations are separately applied on the data sets (*Transformer_V0S2*, *Transformer_86*). Since service $WS3$ expects only one input

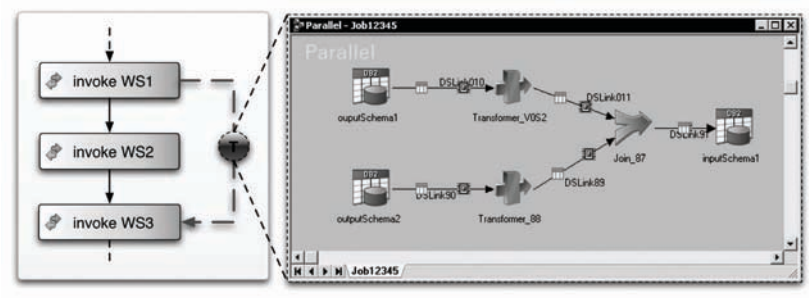


FIGURE 3. ETL Process Inside BPEL Data Transition

data set (*inputSchema1*), the transformed data sets are joined together (*Join_87*). Fundamentally, the following data flow relationships between functional operations exist: $1:1$, $1:N$, $N:M$, $N:1$. For example, a $1:N$ relationship signifies that the output data of a Web service is used as input data for a set of Web services.

The result of the three modeling steps is a process definition with an explicit control flow and with an explicit data flow as well.

3.2. Execution Perspective

In Section 2.2, we reviewed our Data-Grey-Box Web Services. These services offer more information on the data aspect than standard Web services do. As we demonstrated in [8], the usage of Data-Grey-Box Web Services creates an obvious performance benefit in the classic client-server scenario. The composition of Web services with BPEL generates more dependencies. These dependencies are built during the additional modeling phase in the form of data transitions.

Standard Web services do not dispose of the qualification to handle these explicit data flows. Up to now, implicit data flows have been used in BPEL engines, resulting in centralized data flows where the BPEL engine is used as a central data broker. With this principle, BPEL engines do not scale well on data-intensive processes. The additional data aspect information in Data-Grey-Box Web Services now enable us to handle these data transitions with specialized tools. Thereby, every data transition joins the data output reference of the producing service with the data input reference of the consuming services. The data propagation is conducted by a neutral mediator. In the composition scenario, the mediator may now handle the propagation and data transformation as well. The additional transformation execution by the mediator has several advantages: (i) the service does not lose its autonomy, (ii) better load balancing is possible, since the BPEL engine is not responsible any longer for the data transformation with BPELJ or for mediation flows, and (iii) already allocated resources are used, since mediators have to handle the data anyway.

If mediators are used to realize our BPEL data transitions with Data-Grey-Box Web Services, then three service invocation protocols are possible. These

protocols can be categorized as *pessimistic* ones and *optimistic* ones. Pessimistic means that the allocation of the data input reference of the consuming (target) service is done after the producing service has finished. In case of an optimistic approach, the data input references are allocated before the producing service is executed. The resulting three service invocation protocol approaches are described in more detail in the next section. This description is oriented at a small example BPEL process: two Data-Grey-Box Web Services $WS1$ and $WS2$ are connected on the control level and the output data of $WS1$ is used as input data for $WS2$. That means $WS1$ and $WS2$ are connected by a control flow and a data flow.

Pessimistic and Control-Flow-Oriented. With this method, a decentralized data flow is realized in consideration of the control flow. That means that the control flow triggers the execution of the data flows. As a pessimistic approach, the data propagation is not started until it is really needed, that means until the consuming service is started (see Figure 4). Based on our example, WS1 commits its output reference to the BPEL engine. In the invocation procedure of $WS2$, this reference will be transmitted to $WS2$, calling the mediator (the mediation invocation is done by the introduced data layer in the SOAP framework; see Section 2.2 or [8]). The mediator finally transforms the data and delivers them to $WS2$. After the data propagation has been finished, the functional part of $WS2$ can be executed. The advantage of this approach is that data is not transfered over a broker in vain but the data is stored at $WS1$ until $WS2$ triggers the mediator. In the worst case, the duration can amount to hours or days.

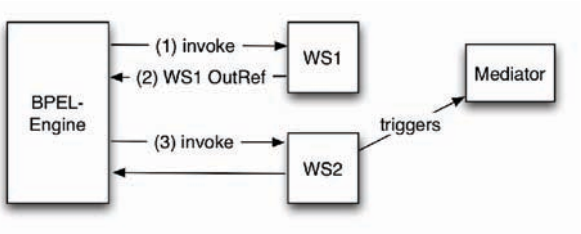


FIGURE 4. Pessimistic and Control-Flow-Bound Service Invocation

Semi-Pessimistic. With this method, centralized as well as decentralized data flows are possible. Figure 5 illustrates the service invocation protocol approach. In this case, the BPEL server pre-defines all input and output references for the participating Web services independent from control flow or data flow information. That means the data is temporarily stored at a third-party site. In the centralized case, the BPEL server must own a data source to temporarily store the data, while in the distributed case, the BPEL server uses arbitrarily distributed data stores as temporary storage devices.

Fundamentally, this service-invocation approach is compatible to the existing procedure (centralized data flow). However, instead of handling the data 'by value',

the data is coordinated by reference. If necessary, the data can be propagated to the BPEL server to get 'by value' access. Disadvantages are (i) the interlocking of the engine with the data flow, and (ii) the fact that two data propagation operations (two mediator calls) are necessary to exchange data between two services. An advantage of this invocation principle is that it is always applicable.

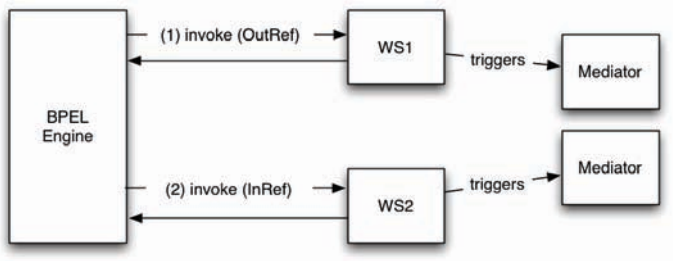


FIGURE 5. Semi-Pessimistic Service Invocation

Optimistic and Data-Flow-Oriented. The third service invocation protocol approach is optimistic. That means the output data is transfered immediately after its creation in $WS1$. Therefore, the input reference of $WS2$ is committed during the invocation of $WS1$. The data propagation through a mediator is done before $WS2$ is invoked. To realize this approach, the invocation protocol has to be changed in general. We introduce a *preinvoke*, which is illustrated in Figure 6. During this *preinvoke*, the data resources for the input data will be allocated in $WS2$. This input reference will be returned to the BPEL engine and used during the invocation of $WS1$. This *preinvoke* allocation implies a policy mechanism to ensure that only authorized invocations are processed. This approach is efficient if Web services are invoked asynchronously. Thus, a parallel control and data flow is possible.

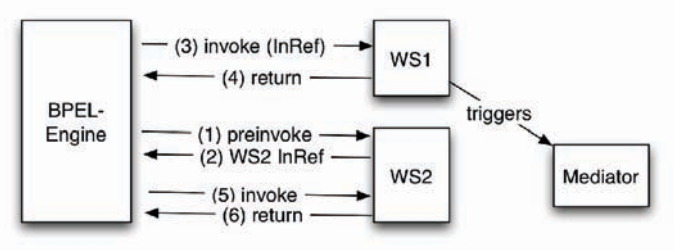


FIGURE 6. Optimistic and Data-Flow-Bound Service Invocation

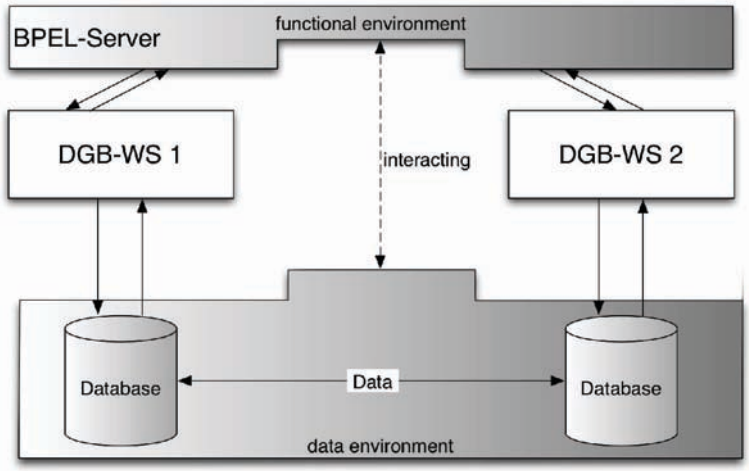


FIGURE 7. Data-Aware SOA Environment

3.3. Summary and Further Investigations

The result of the combination of *Data-Grey-Box Web Services* with our proposed BPEL data transitions is a data-aware SOA environment as illustrated in Figure 7. In such an environment, a functional component (BPEL server) and a data component interact together. Data-Grey-Box Web Services are still loosely coupled as regular Web services. Moreover, the data exchange between participating Web services happens with specialized data propagation tools.

The presented BPEL data transitions with the three service invocation principles create some interesting questions for further research. The additional data-flow modeling phase for data transitions is the starting point of several optimization approaches (see Figure 8). It is desirable to have the possibility to model the data propagation and transformation in an abstract way. From this, it should be possible to choose the best strategy to transfer the data, for example, through ETL, stored procedures or other specialized approaches. Depending on the concrete underlying scenario, several optimized process execution plans can be derived from this abstract process model.

The second point deals with the execution of the process, in particular with the selection of one out of the three presented service invocation principles. The optimistic as well as the pessimistic approach have the disadvantage that one side has to store the data for a long time (in the worst case). The advantage, however, is that in both approaches, only one data propagation operation is required to deliver data from the source Web service to the target Web service, while the semi-pessimistic principle uses two data propagation operations. To tackle this problem from a process perspective, we want to introduce a Data-Grey-Box Web Service policy indicating the time span before and after a certain event during

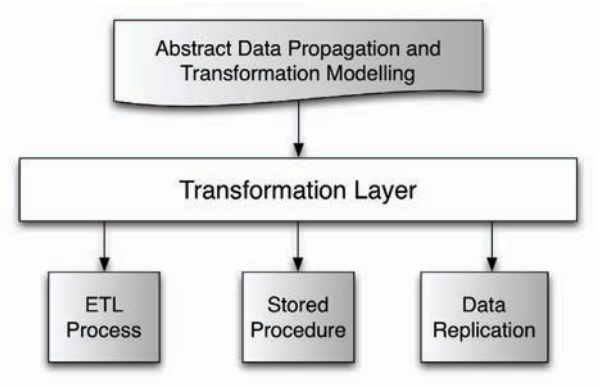


FIGURE 8. Optimization Approach

which data can be stored on the service's side. With the help of some process statistics, we want to determine the best invocation principle. Moreover, some runtime adaptation techniques have to be developed.

4. Implementation Details

In this section, we describe our prototypical realization of the entire concept within the Websphere Integration Developer (WID). The BPEL^{DT} process for our two-phase clustering strategy is depicted in Figure 9. Again, we used three different microarray data sets as starting point. These microarrray data sets are persistently stored in a relational database system (IBM DB2), and *Data-Grey-Box Web Services* allow access to them. The normalization and clustering are realized as single *Data-Grey-Box Web Service* (*getLocalModel* Service). The aggregation of the local clustering result to a global clustering result is done by the data-grey-box Web service *computeGlobalModel*. All data-grey-box Web services use a relational database system in order to allow the efficient and scalable processing of incoming data and some tasks within those services are pushed down to the database system [11].

In contrast to the more general implementation presented in [8], *Data-Grey-Box Web Services* are realized in a slightly different way within WID. The separation of parameters and data in the interface description is done by embedding schema descriptions in the DBM format for input and output data in the *types* section. Through naming conventions, a service requestor is able to determine which database-oriented schemas for input and output data are assigned to each functional operation. Moreover, instead of integrating of the introduced data-layer component within the SOAP framework, each *Data-Grey-Box Web Service* interface includes various administrative operations. These administrative operations are normally hidden from the service requestor by the data-layer component.

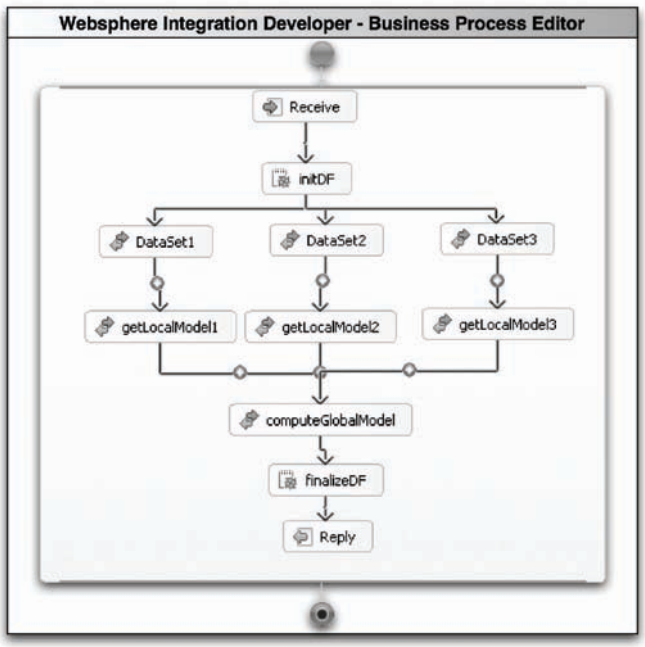


FIGURE 9. Two-Phase Clustering with BPEL^{DT}

BPEL data transitions are prototypically realized using <*flow*>-link types with transition conditions. These transition conditions include the data flow task descriptions. As execution strategy, we use the optimistic and data-flow-oriented approach. We think this strategy offers some advantages with regard to performance compared to the other execution approaches. To enable the optimistic approach, two additional activities are necessary. The first activity, *initDF*, consists of the initialization of all data flow sessions at the participating *Data-Grey-Box Web Services* one th process has been started. That means necessary data resources are allocated at the services. The last activity of the process, *finalizeDF*, closes all data sessions and deallocates all data resources at the services. However, the implementation of the semi-pessimistic strategy is straightforward.

The abstract definition of the data transformation is done with the XML-based Mapping Specification Language (MSL) [15]. The Rational Data Architect (RDA) can be used to specify such transformations based on schema information. For each data flow, an MSL template will be generated containing the data output schema of the data-producing Web service (source schema) and the input schema of the data-consuming service (target schema) (see Figure 10). Those templates have to be refined by the user.

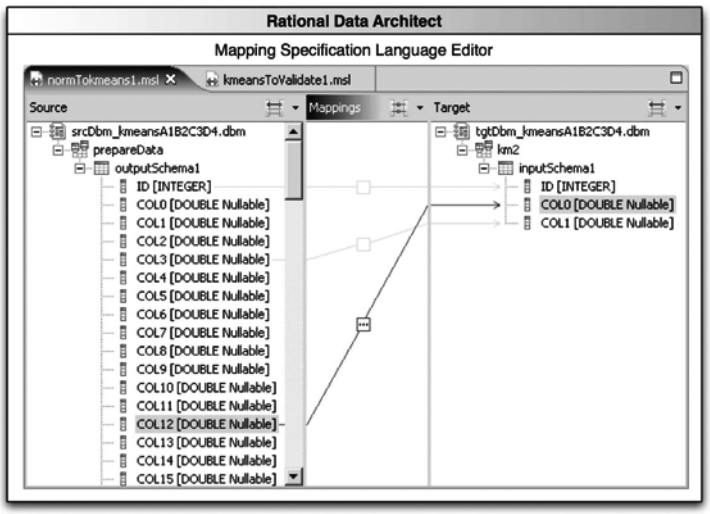


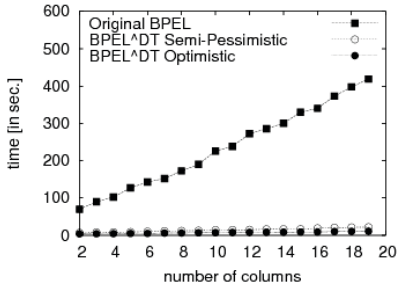
FIGURE 10. MSL Template in Rational Data Architect

The result of the entire modeling part is a functional process definition with explicit data flows containing transformation specifications in MSL form. During the process deployment, the BPEL process definition is investigated and included data flows will be mapped to parametrized DataStage ETL job descriptions. In this mapping step, the MSL specification will be considered. That means the data propagation is restricted to ETL for now.

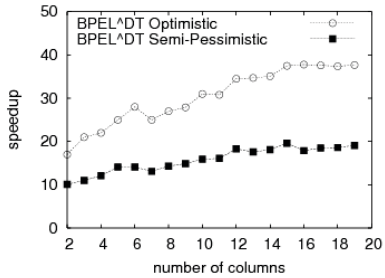
At process runtime, the data flows are executed according to our optimistic and data-flow-oriented strategy. The parametrized DataStage ETL job descriptions are invoked with the current information regarding the data source from our *Data-Grey-Box Web Services*.

5. Evaluation

In this section, we evaluate our proposed BPEL^{DT} approach regarding the performance gain. An evaluation of the data-grey-box Web services is presented in [8]. In this BPEL^{DT} evaluation, we delivered microarray data matrices from a provider service to an analysis service. In the experiment, we measured the time for the data exchange between these two services (ii) with the original BPEL approach, where the BPEL server is the broker, (ii) with BPEL^{DT} with the optimistic execution strategy and (iii) with BPEL^{DT} with the semi-pessimistic approach. The resulting times are depicted in Figure 11(a). In the conducted experiments, we changed the number of columns of the microarray data matrices, whereas the number of rows remained fixed (rows=20,000).



(a) Evaluation Times BPEL and BPEL^{DT}



(b) Speedup of BPEL^{DT}

FIGURE 11. Evaluation of BPEL^{DT}

As illustrated in Figure 11(a), both BPEL^{DT} execution strategies are faster than the original BPEL approach. The data flows in case of BPEL^{DT} are executed with the ETL tool DataStage. The resulting speedups compared to the SOAP transmission in case of the original BPEL approach are depicted in Figure 11(b). As expected, the semi-pessimistic execution strategy is slower than the optimistic as well as the pessimistic approach. In the semi-pessimistic case, an additional data propagation is necessary, since the data is temporarily stored at a third position. Therefore, this strategy is slower than the other two execution approaches. In general, this experiment confirms the evaluation results of our *Data-Grey-Box Web Services*. *Data-Grey-Box Web Services* and BPEL^{DT} are more suitable for data-intensive service applications with regard to performance issues than the original approach.

6. Conclusion

Up to now, the current standards of SOAP, WSDL and BPEL define a 'by value' handling of data in the service-oriented architecture. However, this 'by value' handling is not suitable for data-intensive application. In this paper, we have illustrated our whole service-oriented solution for data-intensive applications. This solution includes *Data-Grey-Box Web Services* [8] which allows the integration of specialized data propagation tools in the invocation process. In this case, the data is not longer handed 'by value'. The main focus of this paper is the introduction of BPEL data transitions to enable an efficient composition and execution of *Data-Grey-Box Web Services*. Aside from theoretical background, we present our prototypical realization, and some evaluation result. Furthermore, we highlight further reseach aspects in this direction.

References

[1] Nayef Abu-Ghazaleh and Michael J. Lewis. Differential deserialization for optimized soap performance. In *Proceedings of the ACM/IEEE SC2005 Conference on High Performance Networking and Computing (SC 2005, November 12-18, 2005, Seattle, WA, USA)*, 2005.

[2] Nayef Abu-Ghazaleh, Michael J. Lewis, and Madhusudhan Govindaraju. Differential serialization for optimized soap performance. In *Proceedings of the 13th International Symposium on High-Performance Distributed Computing (HPDC 2004, 4-6 June, Honolulu, Hawaii, USA)*, pages 55–64, 2004.

[3] Business Process Modeling Notation (BPMN) Information. http://www.bpmn.org/.

[4] Min Cai, Shahram Ghandeharizadeh, Rolfe R. Schmidt, and Saihong Song. A comparison of alternative encoding mechanisms for web services. In *Database and Expert Systems Applications, 13th International Conference*, 2002.

[5] Girish Chafle, Sunil Chandra, Vijay Mann, and Mangala Gowri Nanda. Decentralized orchestration of composite web services. In *Proceedings of the 13th International Conference on World Wide Web-Alternate Track Papers and Posters(WWW 2004, New York, NY, USA, May 17-20)*, page 134143, 2004.

[6] Kenneth Chiu, Madhusudhan Govindaraju, and Randall Bramley. Investigating the limits of soap performance for scientific computing. In *11th IEEE International Symposium on High Performance Distributed Computing*, 2002.

[7] Marc Girardot and Neel Sundaresan. Millau: an encoding format for efficient representation and exchange of xml over the web,http://www9.org./w9cdrom/154/154.html.

[8] Dirk Habich, Steffen Preissler, Wolfgang Lehner, Sebastian Richly, Uwe Assmann, Mike Grasselt, and Albert Maier. Data-grey-box web services in data centric environments. In *Proceedings of the 2007 International Conference on Web Services (ICWS 2007)*, pages 976–983, 2007.

[9] Dirk Habich, Thomas Wächter, Wolfgang Lehner, and Christian Pilarsky. Two-phase clustering strategy for gene expression data sets. In *Proceedings of the 2006 ACM Symposium on Applied Computing - Bioinformatics Track (SAC 2006, Dijon, France, April 23-27)*, pages 145–150, 2006.

[10] Steffen Heinzl, Markus Mathes, Thomas Friese, Matthew Smith, and Bernd Freisleben. Flex-swa: Flexible exchange of binary data based on soap messages with attachments. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, Washington, DC, USA, 2006. IEEE Computer Society.

[11] Alexander Hinneburg, Wolfgang Lehner, and Dirk Habich. Combi-operator: Database support for data mining applications. In *Proc. of 29th International Conference on Very Large Data Bases*, 2003.

[12] IBM. Ibm information server, 2007. http://www-306.ibm.com/software/data/integration/info_server/.

[13] Bettina Kemme and Gustavo Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, 2000.

[14] Albert Maier, Bernhard Mitschang, Frank Leymann, and Dan Wolfson. On combining business process integration and etl technologies. In *Datenbanksysteme in Business, Technologie und Web, 11. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (BTW 2005, Karlsruhe, 2.-4. März)*, pages 533–546, 2005.

[15] Mapping Specification Language. http://www.research.ibm.com/journal/sj/452/roth.html.

[16] Mangala Gowri Nanda, Satish Chandra, and Vivek Sarkar. Decentralizing execution of composite web services. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2004, October 24-28, Vancouver, BC, Canada)*, page 170187, 2004.

[17] Mangala Gowri Nanda and Neeran M. Karnik. Synchronization analysis for decentralizing composite web services. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC03, Melbourne, FL, USA, March 9-12)*, page 407414, 2003.

[18] Alex Ng. Optimising web services performance with table driven xml. In *Proc. of the 17th Australian Software Engineering Conference*, 2006.

[19] Lucian-Mircea Patcas, John Murphy, and Gabriel-Miro Muntean. Middleware support for data-flow distribution in web service composition. In *Proceedings of the combined Doctoral Symposium and 15th PhDOOS Workshop at the 19th European Conference on Object Oriented Programming(PhDOSS, Glasgow, Scotland, July 25)*, 2005.

[20] Specification of BPEL. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[21] Specification of BPELJ. http://www-128.ibm.com/developerworks/library/specification/ws-bpelj/.

[22] Robert van Engelen. Pushing the soap envelope with web services for scientific computing. In *Proc. of the International Conference on Web Services (ICWS'03)*, 2003.

[23] Panos Vassiliadis, Alkis Simitsis, and Spiros Skiadopoulos. Conceptual modeling for etl processes. In *Proc. of the 5th ACM international workshop on Data Warehousing and OLAP*, pages 14–21, New York, NY, USA, 2002. ACM Press.

[24] Patrick Widener, Greg Eisenhauer, and Karsten Schwan. Open metadata formats: Efficient xml-based communication for high performance computing. In *10th IEEE International Symposium on High Performance Distributed Computing*, 2001.

Dirk Habich
Dresden University of Technology, Germany
Database Technology Group
e-mail: `dirk.habich@tu-dresden.de`

Sebastian Richly
Dresden University of Technology, Germany
Software Engineering Group
e-mail: `sebastian.richly@inf.tu-dresden.de`

Steffen Preissler
Dresden University of Technology, Germany
Database Technology Group
e-mail: steffen.preissler@tu-dresden.de

Mike Grasselt
IBM Boeblingen Lab, Germany
Information Server SWG SOA Integration
e-mail: grasselt@de.ibm.com

Wolfgang Lehner
Dresden University of Technology, Germany
Database Technology Group
e-mail: wolfgang.lehner@tu-dresden.de

Albert Maier
IBM Boeblingen Lab, Germany
Information Server SWG SOA Integration
e-mail: amaier@de.ibm.com

Whitestein Series in Software Agent Technologies, 129–140

Towards Resource-Oriented BPEL

Hagen Overdick

Abstract. Service orientation is the de-facto architectural style, today. But, what actually is a service and how should service boundaries be chosen? Resource orientation, once seen as a "light-weight" approach to Web services, is reshaping itself as a modeling strategy to service orientation. Along comes the realization that resources are in-fact complex state machines. Currently, there is no accepted standard for modeling the internal state of resources. In this paper, BPEL is proposed as a modeling language for resources and necessary extensions to BPEL are outlined.

Keywords. BPEL,SOA,REST,BPM.

1. Introduction

There has been a lot of discussion about service-oriented architectures (SOA) [1], lately. A service is a mechanism to enable access to one or more capabilities. The eventual consumers of the service may not be known to the service provider and *may demonstrate uses of the service beyond the scope originally conceived by the provider* [2]. If a provider may not know the actual use of a service, what makes a service a service? What minimum level of functionality must a service provide to be called a service? Furthermore, according to recent studies [3], about two-thirds of all services deployed today are data-centric. Is a memory cell already a service? Resource orientation [4] solves this dilemma by making every entity explicit, not just services. Such explicit entity is called a resource. If one can find a noun for an entity, it qualifies as a potential resource. All properties of services still hold true for resources, i.e. they have an independent life- cycle, a globally unique reference, and their interaction style is stateless message exchange.

Resource orientation is the dominant architectural style on the Internet, as it is the scientific foundation of the World Wide Web [5]. Resources have a globally shared request message classification system, confusingly called *uniform interface*. The idea is, that even without semantic understanding of the messages exchanged, the classification provides additional benefits to the overall architecture. However,

up to now, the World Wide Web favors an informal, ad-hoc description of complex resource behaviors. Roy Fielding coined the term "hypertext as the engine of application state" [4], upgrading this ad-hoc fashion from bug to feature; quite a successful feature indeed measured by the success of the World Wide Web itself.

Enterprises and research on the other hand are very much interested in the description of complex behaviors. Out of this need, Web Services [6] were created as an additional layer on top of resource orientation, including the Web Service Definition Language (WSDL) [7] to describe service interfaces and the Business Process Execution Language (BPEL) [8] for behavioral descriptions. Recently, resource orientation is rediscovered as a viable subset of service orientation [9]. This also raises the question, if complex resource behavior can be expressed formally. With the introduction of the Web Application Definition Language (WADL) [10], a candidate for the description of interfaces is given. This paper outlines how BPEL can be adapted to describing process aspects of resources.

The remainder of this paper is structured as follows: In chapter 2, an introduction to resource orientation is given. In chapter 3, an example of a complex resource behavior is shown to illustrate the requirements of a resource-oriented process language. In chapter 4, BPEL is introduced as a viable candidate for such a language and the necessary extensions are outlined. Chapter 5 discusses related work and chapter 6 concludes this paper with a summary and outlook.

2. Resource Orientation

Resource orientation is a subset of service orientation. As such, it can be regarded as a modeling strategy for services. Instead of a few "gateway" services, with carefully crafted custom interfaces, all entities of the modelled system expose a uniform interface. To illustrate this, let us look at an example of such uniform interface. The Hypertext Transfer Protocol (HTTP) [11] defines its uniform interface for requests as:

GET. : Messages labeled as GET have an empty service request and are guaranteed to have no substancial effect within the receiver of such request, i.e. they are *safe* to call. GET responses are expected to be a description of the current state of the targeted resource. These attributes allow GET to act as a universal reflection mechanism, it can be issued without any prior knowledge of the resource. Also, as GET does not alter the state of the targeted resource, the response can be cached. This has great benefits to a distributed architecture and both aspects can be seized without prior semantic knowledge of the targeted resources. In the physical world, GET request can be correlated to looking.

PUT. Messages labeled as PUT do cause an effect in the targeted resource, but do so in an *idempotent* fashion. An idempotent interaction is defined as replayable, i.e. the effect of N messages is the same as that of 1. In a distributed system, where transactions may not be available, this is a great help for error recovery as idempotent messages can be delivered at least once without any effort, just retry

until acknowledged. Again, thanks to the uniform interface, the assumption of idempotency can be made without any prior semantic knowledge of the resource involved. In the physical world, this correlates to physical interaction, although replaying the exact same "message" is only a theoretical mind exercise.
DELETE. Messages labeled as DELETE do cause an effect in the targeted resource, where that effect is expected to be a termination. Just as PUT, DELETE is defined as *idempotent*. However, as with all messages, the interpretation is solely the responsibility of the receiver, i.e. a DELETE has to be regarded as "please terminate". In the physical world, this correlates to sending a notice of cancelation.
POST. All other types of messages are labeled as POST, i.e. they cause an effect in the receiver and they are not safe to replay. This is a catch all mechanism for all messages that can not be described by the prior verbs. Without a uniform interface, all messages would be treated like this, loosing context free reflection, caching and replayability.

The uniform interface tries to lower the barrier of entry to a client and it also includes a characterization of response messages. Thus, interaction with a resource can start purely on the basis of semantic understanding of the uniform interface. If one obtains a resource identifier, the uniform interface provides a minimum level of shared semantics to start with. However, this set of semantics is not restricted. A uniform interface simply enforces that any label (or verb as HTTP calles them) may be applied to *all* resources. To increase the likelihood of understanding, both client and resource perform content type negotiation on each request. Content type negotiation honors the fact that there are many ways of encoding information.

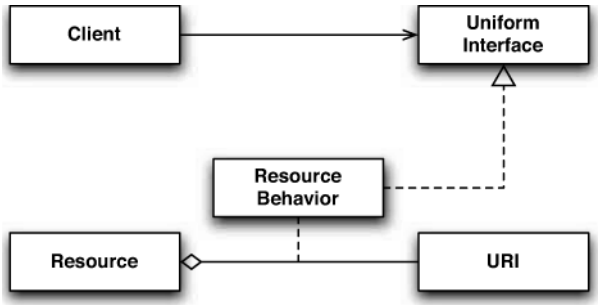


FIGURE 1. Exposing behavior in resource-oriented architectures

Conceptually, a resource is beyond the communication system, i.e. a client can only communicate with it via the uniform interface it addresses by a globally unique identifier, e.g. a URI [12]. The relationship between a resource and a URI may change over time. Yet, resource orientation today spends very little effort on describing the *underlying process* defining the relationship between a resource

and its URIs. While a URI is bound to a resource, it exposes a certain resource behavior. In Figure 1 the relationship between these concepts described is shown.

3. Example of a complex resource behavior

As an illustration, let us now introduce a complex resource most people should be familiar with: an online ordering process. In Figure 2, a very simple version is illustrated. A shopping cart is created by the user by adding an initial item. Adding items can be repeated as many times as the user likes. If the user simply stops interacting with the shopping cart, it may time out or the user decided to check out by choosing a payment method. At this point the user is presented with the content of the shopping cart, the chosen payment method, and the total bill to confirm before actually committing the order.

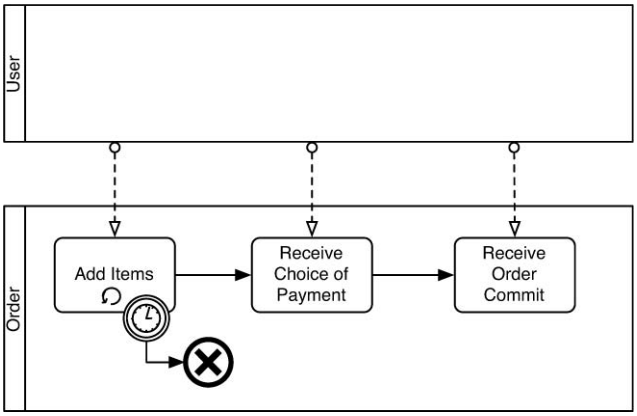


FIGURE 2. Shopping cart as a complex resource

The first step towards a good resource-oriented design is to identify the relevant resources. Is a shopping cart actually a resource on its own, or just a state of an order? By modeling the later—the shopping cart to be just a state of an order resource—we can uniquely identify the order in all stages, e.g. shopping cart, check-out, assembly, in-delivery, and post-delivery. The user is given a single URI, something to bookmark in a browser. Clicking on such a bookmark will issue a GET request. A GET request in a resource-oriented view is nothing else than introspecting the current state of a resource. In the true spirit of *hypermedia as the engine of application state* the returned representation of the current resource should include all relevant links and possible interactions. By doing so, the client is never forced to understand the process as such, being able to browse and post is the only requirement to participate as a client. This simplicity is the true strength of a resource-oriented design and the foundation of the World Wide Web's success story. At the same time, this motivates the resistance against formal descriptions

of interfaces and processes as practiced in a Web services environment. While resource orientation does not conflict with formal interface descriptions and in fact would benefit from it, any attempt to introduce such formalism to resource orientation must honor the fact that resource orientation can and will work without such formalism.

Nevertheless, it should have become apparent that resources are indeed complex state machine and that such state machines can be expressed as processes, matching the business concepts used to motivate the system in the first place. We already identified a shopping cart to be just a state or dependent sub-resource of an order. However, this opens the question of how to choose resource boundaries. Is the order a resource in itself or is it a sub-resource of the store? In [13] a very pragmatic answer is given: Breaking down an application into as many resources as possible benefits scalability and flexibility, but at the same time the resource is the scope of serializability, i.e. there may not be transactions across resource boundaries. I.e. the order is not dependent on the store (at least in a transactional view), but the order items probably are dependent on the order, as an order item may only be changed as long as the order has not been committed.

Before we outline a process notation in the next chapter, let us summarize our findings: A resource may consist of several complex states, each able to expose a set of URIs. Each of these URIs expose a certain behavior of the resource. Interaction with any of the resource's URIs is classified into *safe* (one interaction has the same effect as zero interactions), *idempotent* (one interaction has the same effect as n interactions), or unrestricted, i.e. such interaction is able to produce an uncontrollable side-effect and/or change the internal state of the resource. Also, a resource must be able to extract URIs from representations received via interaction and be able to then interact with the extracted URIs, as this is a fundamental aspect of resource orientation.

4. Resource-oriented BPEL

In this section, BPEL is introduced and extensions for modeling complex resources are outlined.

4.1. BPEL

BPEL is arguably the de facto standard for specifying processes in a Web services environment. BPEL provides *structured activities* that allow the description of the control flow between the *interaction activities*. BPEL does not support explicit data flow, but rather relies on shared variables referenced and accessed by interaction activities and manipulation activities. The control flow between activities can be structured either block-based by nesting structured activities like `<sequence>` and `<flow>`, or graph-based by defining directed edges (called `<links>`) between activities inside `<flow>` activities. Both styles can be used as the same time, making BPEL a hybrid language.

Beyond control flow and data manipulation, BPEL also supports the notion of scopes and allow for compensation handlers and fault handlers to be defined for specific scopes. Hence, scopes represent units of works with compensation-based recovery semantics. Fault handlers define how to proceed when faults occur, compensation handlers define how to compensate already completed activities, as processes not transactional and consequently must be rolled back explicitly. Further more, scopes allow for event handlers which can be regarded as repeatable, attached sub-processes [14] triggered by events.

4.2. BPEL without Web Services

The wide-spread acceptance and the sophistication of the control flow constructs, make BPEL a strong candidate when trying to formally express the process governing the relationship between a resource and its URIs. Both the interaction activities and the grouping mechanism that allows modeling complex message exchanges depend on WSDL. However, in [15] BPEL light is introduced, a WSDL-less version of BPEL. While BPEL light itself still is not a good match for resource orientation, a clear path on how to remove the dependency on WSDL from BPEL and adding new interaction models in a compatible way is shown clearly. In essence, the elements `<receive>`, `<reply>`, `<invoke>`, `<onMessage>` within a `<pick>`, as well as `<onEvent>` within an `<eventHandler>` need to be replaced by constructs not relying on WSDL.

4.3. Using BPEL to model resource states

BPEL does not have an explicit state modeling, but an implicit via the `<scope>` construct. Generally speaking, a POST message or an event may cause a state transition. However, while in a state, as many GET, PUT, and/or DELETE messages may arrive, as they are safe and/or idempotent.

As shown in Figure 3, BPEL provides the concept of event handlers to model GET, PUT, and DELETE interaction as attached, repeatable subprocesses. Enforcing *safe* and *idempotent* characteristics of those interactions is beyond the scope of this paper. However, a straight forward solution may be disallowing write access to any variable during a GET interaction to ensure safeness. PUT and DELETE can be enforced idempotent by disallowing write access to any variable read, i.e. overwriting a variable is ok, computing a new value based on the old one is not. Such interaction may be executed several times and in parallel, while POST interaction or events move the BPEL process into a new scope.

4.4. Resource interaction in BPEL

Web services try to abstract from the communication protocol, providing support for a wide range of interaction models, such as asynchronous or one-way interaction. Resource orientation on the other hand puts much effort into the core protocol as the lowest level of shared semantics. The dominate resource-oriented protocol is HTTP. Consequently there is no point in abstracting away from it when modeling

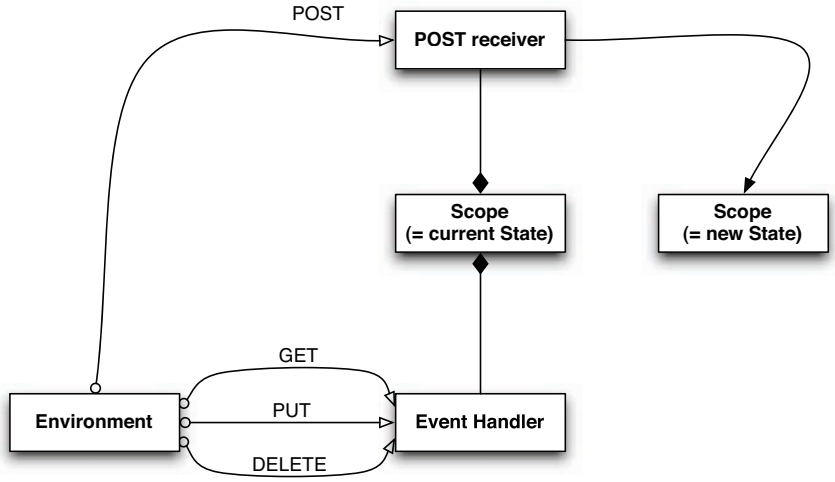


FIGURE 3. Using BPEL to model resource states

interaction in BPEL. In fact many proponents of resource orientation have major concerns with any attempt to hide the protocol layer behind an abstraction.

All interaction in HTTP is based on synchronous request-response. Asynchronous communication is supported by identifying either the asynchronous sender or receiver by an explicit URI and sending it along in the initial request. I.e. at the protocol level, there will be a synchronous request and then an independent synchronous response push or pull. This design makes the interaction much simpler, but requires a simple mechanism to construct URIs. There is currently one attempt to standardized URI templating [16] applicable to creation, matching, and selection of URIs. Within WADL, URI templates are already used for matching and selection of URIs. To a resource itself, creation of URIs must be available, too. Coming back to our example process, upon receiving a shopping item, it must be added to the shopping cart, in turn generating one or more URIs for the newly created item.

```
<assign>
  <copy>
    <from>rbpel:generate-uri("./item/{itemNumber}")</from>
    <to variable="newItemURI" />
  </copy>
</assign>
```

FIGURE 4. URI creation by XPath-method

The easiest way to provide such functionality is to offer an XPath function. Figure 4 shows how the regular `<assign>` construct is used to create a new URI

using such XPath function. URIs themselves do not need a special construct and can be kept in normal variables.

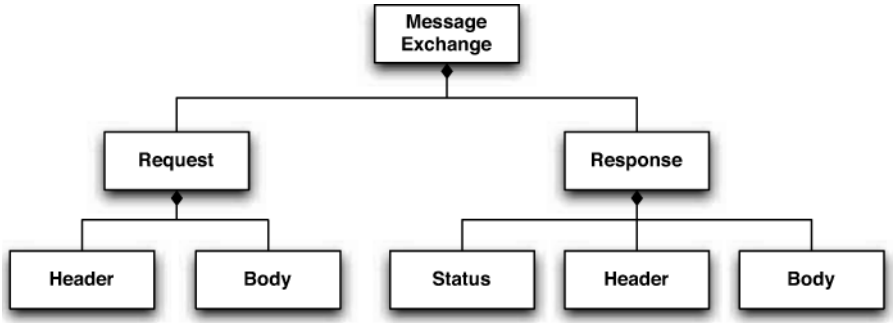


FIGURE 5. URI interaction

With URIs introduced to BPEL, let us look at URI interaction again, as shown in Figure 5. Any URI interaction is synchronous and the tuple of *request* and *response* is grouped into a *message exchange*. Both request and response contain a header and a body, where the header includes the content type of the body. The response also includes a *status*, which is part of the uniform interface of HTTP and encodes a general indication of how the request was processed.

Remember, a message exchange is always synchronous. This reduces the possible interaction patterns to *send-receive* and *receive-reply*. While it is tempting to simplify the BPEL constructs into `<send>` and `<receive>` elements with the complete handling of the request as child elements, the BPEL specification does not appear to allow extension activities with child elements, hence we refrain from doing so and stick with the tradtional `<send>`, `<receive>`, and `<reply>` constructs without children. However, the newly introduced activities all have a *messageExchange* attribute by which the required data structures—as shown in Figure 5 are referenced.

In Figure 6 the fragment from Figure 4 is completed to a complete `<onMessage>` block. Notice the *path* attribute containing a relative URI template. The given template is relative to the BPEL process, as each instance of the process is assigned a URI itself. The exact details of the message the `<onMessage>` activity is waiting for is described by reusing the `<method>` element of WADL [10]. Here, the only criteria is that the message is send as a POST. WADL itself is quite descriptive and this descriptive power can be used to model pattern matching on request, i.e. several `<onMessage>` activities waiting on the same URIs with the same verb but different contents. The `<reply>` activity again references the *messageExchange* data structure by attribute. Here, some convenience elements are shown (`<status>` and `<param>`), their functionality could be simply mapped to `<assign>` working on the data structure. However, this fragment shows how a

```
<rbpel:onMessage path="./item/new" messageExchange="createItem">
  <wadl:method name="POST" />
  <sequence>
    <assign>
      <copy>
        <from>rbpel:generate-uri("./item/{itemNumber}")</from>
        <to variable="newItemURI" />
      </copy>
    </assign>
    <rbpel:reply messageExchange="createItem">
      <rbpel:status>201</rbpel:status>
      <rbpel:param name="Location" style="header">$newItemURI</rbpel:param>
    </rbpel:reply>
  </sequence>
</rbpel:onMessage>
```

FIGURE 6. Creating a shopping cart item

new URI is generated by template and returned to the requester in the *Location Header* as outlined in the HTTP specification.

The complete BPEL for all functionality hidden in the *Add Items* activity of Figure 2 is shown in Figure 7.

The loop—depicted by a curved arrow on the "Add Items" activity in Figure 2–is mapped to a `<repeatUntil>` block. Upon receive a POST to the *checkout* URI the loop is left by settint the *commitRequest* variable to *true*. Also, the new internal state of the resource modeled by BPEL process outlined has a URI by itself. The requesting client is redirect to that URI by issuing a 303 status, again as outlined by the HTTP specification.

5. Related work

There are many other language available as a foundation to modeling resource behavior, such as Web Service Choreography Interface (WSCI) [17] or the Web Service Conversation Language (WSCL) [18]. However, mind share is vital to language selection and BPEL seems to be able to form a common ground for various interest groups. Also, even though some constructs my be expressed more elegantly, BPEL is designed as an open, extensible language laying a clear track of how to integrate the required functionality, as shown in the course of this paper. Describing static resource interfaces, the author is unaware of any alternative to the Web Application Description Language. On the other hand, WADL can be seen as a mashup of HTTP, RelaxNG [19], and XML Schema [20], so these standards should be mentioned here as well.

```
<repeatUntil>
  <scope
      xmlns:rbpel="http://bpt.hpi.uni-potsdam.de/ns/rbpel"
      xmlns:wadl="http://research.sun.com/wadl/2006/10">
    <eventHandlers>
      <rbpel:onMessage path="./item/{itemNumber}" messageExchange="itemShow">
        <wadl:method name="GET" />
        <!-- return representation of item $itemShow.itemNumber -->
      </rbpel:onMessage>
      <rbpel:onMessage path="./item/{itemNumber}" messageExchange="itemUpdate">
        <wadl:method name="PUT" />
        <!-- update and return item $itemUpdate.itemNumber -->
      </rbpel:onMessage>
      <rbpel:onMessage path="./item/{itemNumber}" messageExchange="itemDelete">
        <!-- delete item $itemDelete.itemNumber -->
      </rbpel:onMessage>
      <rbpel:onMessage path="./item/new" messageExchange="createItem">
        <wadl:method name="POST" />
        <sequence>
          <assign>
            <copy>
              <from>rbpel:generate-uri("./item/{itemNumber}")</from>
              <to variable="newItemURI" />
            </copy>
          </assign>
          <rbpel:reply messageExchange="createItem">
            <rbpel:status>201</rbpel:status>
            <rbpel:param name="Location" style="header">$newItemURI</rbpel:param>
          </rbpel:reply>
        </sequence>
      </rbpel:onMessage>
    </eventHandlers>
    <pick>
      <rbpel:onMessage path="./checkout" messageExchange="transfer_to_payment">
        <wadl:method href="/wadl/post/method/definition" />
        <sequence>
          <assign>
            <copy>
              <from>true</from>
              <to>$commitRequest</to>
            </copy>
          </assign>
          <rbpel:reply messageExchange="transfer_to_payment">
            <rbpel:status>303</rbpel:status>
            <rbpel:param name="Location" style="header">"./checkout"</rbpel:param>
          </rbpel:reply>
        </sequence>
      </rbpel:onMessage>
      <onAlarm>
        <for>'2h'</for>
        <exit/>
      </onAlarm>
    </pick>
  </scope>
  <condition>$commitRequest</condition>
</repeatUntil>
```

FIGURE 7. Complete example of "Add Items" activity

6. Summary and Outlook

In the course of this paper, resource orientation was introduced as a viable subset of service orientation. Resource as such are complex state machines, exposing one or more uniform interfaces over time. This can be formally expressed as a complex state machine. The main contribution of this paper is to identify BPEL as a suited candidate for modeling such state machines and the necessary modifications to BPEL were outlined. All of these modifications are in the scope of the extension mechanisms of the BPEL specification.

The next steps will involve a exact specification of the extensions outlined and a reference implementation, possibly building upon an existing BPEL engine. This possibility is one of the strong arguments for using BPEL along with the already strong mind share of the BPEL community.

At the same time, a resource-oriented BPEL can be the foundation for a next-generation web framework centering around process models as the core artefact of application design.

References

[1] Burbeck, S.: The tao of e-business services (2000) http://www-128.ibm.com/developerworks/library/ws-tao/.

[2] Matthew, C., Laskey, K., McCabe, F., Brown, P.F., Metz, R.: Reference Model for Service Oriented Architecture 1.0. Technical Report Committee Specification 1, OASIS Open (2006) http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

[3] Gardner, D.: Soa wikis, soa for saas, and the future of business applications. Technical report, Interarbor Solutions (2007) http://blogs.zdnet.com/Gardner/?p=2395.

[4] Fielding, R.T.: Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine (2000) Chair-Richard N. Taylor, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[5] Berners-Lee, T.: Www: Past, present, and future. IEEE Computer **29** (1996) 69–77

[6] IBM: Web services architecture overview (2000) http://www-128.ibm.com/developerworks/webservices/library/w-ovr/.

[7] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web services description language (wsdl) 1.1. Technical report, W3C (2001) http://www.w3.org/TR/wsdl.

[8] Jordan, D., Evdemon, J.: Oasis web services business process execution language (wsbpel) (2007) http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[9] Overdick, H.: The resource-oriented architecture. In: 2007 IEEE Congress on Services (Services 2007). (2007) 340–347 http://doi.ieeecomputersociety.org/10.1109/SERVICES.2007.66.

[10] Hadley, M.: Web application description language (2006) https://wadl.dev.java.net/.

[11] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol – http/1.1. Technical report, The Internet Engineering Task Force (1999) http://www.ietf.org/rfc/rfc2616.

[12] T.Berners-Lee, R.Fielding, L.: Uniform resource identifiers (uri): Generic syntax. Technical report, The Internet Engineering Task Force (1998) http://www.ietf.org/rfc/rfc2396.txt.

[13] Helland, P.: Life beyond distributed transactions: an apostate's opinion. In: Third Biennial Conference on Innovative Data Systems Research. (2007) http://www-db.cs.wisc.edu/cidr/cidr2007/papers/cidr07p15.pdf.

[14] Großkopf, A.: xbpmn. formal control flow specification of a bpmn-based process execution language. Master's thesis, Hasso Plattner Institut and SAP Research Brisbane (2007)

[15] Nitzsche, J., van Lessen, T., Karastoyanova, D., Leymann, F.: Bpel light. In: 5th International Conference on Business Process Management (BPM 2007), Springer (2007)

[16] Gregorio, J., Hadley, M., Nottingham, M., Orchard, D.: Uri template. Technical report, IETF (2008) http://bitworking.org/projects/URI-Templates/.

[17] Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., Zimek, S.: Web service choreography interface (wsci). Technical report, W3C (2002)

[18] Banerji, A., Bartolini, C., Beringer, D., Chopella, V., Govindarajan, K., Karp, A., Kuno, H., Lemon, M., Pogossiants, G., Sharma, S., Williams, S.: Web services conversation language (wscl). Technical report, W3C (2002)

[19] Clark, J., Makoto, M.: Relax ng specification. Technical report, OASIS Open (2001)

[20] Thompson, H.S., Sperberg-McQueen, C.M., Gao, S., Mendelsohn, N., Beech, D., Maloney, M.: Xml schema 1.1. Technical report, W3C (2006)

Hagen Overdick
Research School
Hasso Plattner Institute for IT Systems Engineering
at the University of Potsdam
D-14482 Potsdam, Germany
e-mail: hagen.overdick@hpi.uni-potsdam.de

Whitestein Series in Software Agent Technologies, 141–157

SSL-over-SOAP: Towards a Token-based Key Establishment Framework for Web Services

Sebastian Gajek, Lijun Liao, Bodo Möller and Jörg Schwenk

Abstract. Key establishment is essential for many applications of cryptography. Its purpose is to negotiate keys for other cryptographic schemes, usually for encryption and authentication. In a web services context, WS-SecureConversation has been specified to make use of negotiated keys. The most popular key establishment scheme in the Internet is the (handshake protocol of the) Secure Socket Layer or Transport Layer Security protocol (SSL/TLS). However, SSL/TLS has primarily been designed to secure HTTP, by encrypting and authenticating TCP connections. It is thus not usable to negotiate keys in SOAP connections with intermediaries. We propose SSL-over-SOAP, a family of key establishment protocols for Web services. It is based the design of the SSL handshake, so security analysis results for standard SSL/TLS apply to our new proposal. We have implemented this protocol in the framework of WS-Trust and WS-SecureConversation.

Keywords: Web Services Security, Key Establishment Protocol

1. Introduction

1.1. Motivation

Security is important for any distributed computing environment: Many passive and active attacks have been described against such systems. Particularly challenging are service-oriented environments where the architecture is implemented based on a range of technologies, and where applications are created as loosely coupled and interoperable services. The Internet and its underlying infrastructure is the most pervasive IT system ever built—accordingly, more and more applications are implemented as Web services. Thus, preserving the privacy and integrity of these messages in service-oriented architectures becomes a challenging part of business integration, and secure message exchange a requirement for the proliferation of Web services.

Because of the complexity of XML based security standards, the well-known *Secure Socket Layer (SSL)* or *Transport Layer Security (TLS)*[1] protocol has become a de facto security standard for Web services. SSL provides a protected TCP channel that can be used by higher order protocols, such as e.g. SOAP over HTTP for Web Services. Because of its eminent role for the Web (as well as for other Internet protocols), the SSL protocol has been examined intensely [13–15], without finding any severe security flaws.

However, classical SSL has some shortcomings when deployed in the context of Web Services. First, SSL is a point-to-point security protocol. Web Services, by contrast, are loosely coupled applications: that is, messages may pass through multiple intermediary nodes, and the bindings to service endpoints may change. In order to establish a secure SSL channel between two service endpoints, each intermediary connection must be protected by SSL, and the application must be able to decide which of the intermediary SSL certificates are trustworthy. Second, SSL is a transport layer security protocol: SSL-protected messages are secured while in transit on the network; after reception, the message plaintext (as recovered by the SSL layer) is forwarded to the application logic. Third, SSL is not aware of the message structure, so messages are protected in an all-or-nothing fashion. Higher layers do not directly benefit from SSL session keys. One benefit of XML security technologies, in contrast, is to provide element-wise signing and encryption: intermediaries can read and alter information only as they are permitted to.

1.2. Motivating Example

Consider an example where a business flow requires passing an invoice through multiple parties. The invoice contains some vital information that only the ultimate receiver is allowed read; however, certain parts of the invoice are to be processed by intermediary parties. SSL fails because intermediaries have access to the complete invoice in plaintext (or they would not be able to examine the invoice at all). By contrast, SSL-over-SOAP allows for establishing a session key between sender and ultimate receiver. This key can be used to authenticate and encrypt the invoice information to protect it from intermediaries. The sender can choose which pieces of information to encrypt.

1.3. Contribution

The WS-* family of security schemes [1] aims to provide a security framework that addresses all the security issues around web services. In particular, WS-SecureConversation [2] defines how to use session keys in WS-Security, but does not specify any specific key exchange protocols.

In this work, we close this gap by re-specifying the SSL handshake protocol and the SSL record layer at the SOAP level, creating a new cryptographic protocol to be used with WS-SecureConversation. We thus do not need the "classical" SSL

[1] TLS is the official name for the more recent protocol versions in the SSL/TLS family. We use the traditional name SSL as an umbrella term since our ultimate goal is to move the protocol ideas away from the transport layer.

at TCP level any more. Instead, we are able to provide all security services offered by SSL (confidentiality, authentication, security of key establishment) at SOAP level.

Incorporating the practically proven SSL protocol technology into the WS-* family of security scheme allows us to design a protocol framework that benefits from both technologies.

The main contribution of the SSL protocol to the web services world is *security*. In SSL, key agreement and authentication are closely connected, and explicit key confirmation is provided by the `Finished` messages at the end of the handshake protocol. By contrast, it is easy to show the the authenticated variant of the Diffie-Hellman key exchange [7] is vulnerable to man-in-the-middle attacks in combination with XML wrapping attacks [12].

We propose SSL-over-SOAP as the first member of a family of practical Web Services key establishment protocols. SSL-over-SOAP provides sufficient protocol flexibility for the security requirements of today's business models. As a first step, we have implemented the following:

- The SSL-over-SOAP handshake protocol is a key transport protocol based on X.509 binary security tokens. It is implemented in the WS-Trust framework.
- The SSL-over-SOAP record layer protects the complete body of SOAP messages, and the `Finished` messages. It provides confidentiality (XML encryption) and authentication (HMAC from XML signature) within the framework of WS-SecureConversation.

1.4. Related Work

Hada and Maruyam [9] propose a session authentication protocol for Web Services. Although they consider the aspect of session resumption, they do not design a key establishment scheme. Follow-up work by Zhang and Xu [18] similarly does not regard key establishment. Herzberg [10] introduces the secure XML transport protocol (SeXTP), which is a ping pong protocol based on XML Encryption and XML Signature. The work does not fit into the Web Services terminology, as it dates back from a time when the WS-standards were in pending state. Although the author illustrates that present XML security standards are capable to negotiate a shared secret, [10] only mentions the Diffie-Hellman key exchange. In contrast to this, our protocol provides a framework for a wide variety of different algorithms and authentication mechanisms, and is open to extensions. Fang et al. [8] have implemented the AuthA protocol for Web services. AuthA is a password-based authenticated key exchange protocol. This protocol is restricted to the use of passwords. By contrast, SSL-over-SOAP provides protocol flexibility. That is, SSL-over-SOAP captures the requirements of different security models and allows the use of modular authentication mechanisms, such as passwords, digital certificates, or Kerberos ticket.

1.5. Organization

The paper is structured as follows. We shortly review the relevant Web Services technologies in Section 2. Then, we present our proposal by first formulating SSL in terms of SOAP message exchanges in Section 3, and subsequently describing a concrete instantiation of this framework in Section 4. We discuss the protocol's security in Section 5. Finally, we conclude our work in Section 6.

2. WS-* Building Blocks

2.1. Notation

We use the following XML syntax style:

- Instead of writing an element <AAA></AAA>, we drop the tag from the closing bracket and write <AAA></> or <AAA/>.
- When writing an element that spans several lines, we rely on indentation to delimit the body, omitting the closing bracket. For example, <AAA><BBB></BBB></AAA> is written as

```
<AAA>
 <BBB />
```

- We omit the namespace definition in the messages and use the following prefixes:

Prefix	Namespace
ds	http://www.w3.org/2000/09/xmldsig#
soap	http://schemas.xmlsoap.org/soap/envelope/
tls	http://www.example.org/tls#
wsse	http://docs.oasis-open.org/wss/2004/01/ oasis-200401-wss-wssecurity-secext-1.0.xsd
wst	http://schemas.xmlsoap.org/ws/2005/02/trust
wsc	http://schemas.xmlsoap.org/ws/2005/02/sc
wsu	http://docs.oasis-open.org/wss/2004/01/ oasis-200401-wss-wssecurity-utility-1.0.xsd
xenc	http://www.w3.org/2001/04/xmlenc#

2.2. SOAP and WS-Security

SOAP is a mechanism for inter-application communication between systems across the Internet, where system implementations can be written in arbitrary languages. SOAP messages are in XML to allow the exchange of structured information.

WS-Security [3] describes how to use XML Signature [17], XML Encryption [16], and security tokens in SOAP messages. (Note that [17] specifies the use of symmetric-key authentication, not just public-key digital signatures: The term "signature" or "digital signature" is extended to cover symmetric authentication.) To this purpose, WS-Security defines a <Security> element to be added to the

SOAP header as a container for all security related content. For WS-Security, it is strongly recommended to identify signed elements via ID attributes (*not* via XPath expressions). A typical WS-Security message is as follows:

```
<soap:Envelope>
  <soap:Header>
    <wsse:Security/>
  <soap:Body>
```

2.3. WS-SecureConversation

WS-SecureConversation (the Web Services Secure Conversation Language) specifies secure communication between services. It defines the message structure for establishing and sharing security contexts, and for deriving keys from security contexts. As with WS-Security, WS-SecureConversation is only a building block and does not provide a complete security solution.

The core element of WS-SecureConversation is the <wsc:SecurityContextToken> element. It consists of the mandatory child element <wsc:Identifier> and several optional elements. The security context is addressed by a UUID specified in <wsc:Identifier>.

2.4. WS-Trust

WS-Trust enables applications to construct trusted SOAP message exchanges, which is determined by security tokens. A typical security token request consists of:

```
<wst:RequestSecurityToken Context? ...>
  <wst:TokenType/>?
  <wst:RequestType/>
  ...
```

The optional element <wst:TokenType> describes the requested token type. The mandatory element <wst:RequestType> specifies the class of function. It allows to add additional elements for the special purpose. The <wst:RequestSecurityTokenResponse> specifies the response to a security token request and is used to retrieve a security token.

3. SSL-over-SOAP Framework

3.1. Design Goals

We lift the SSL framework from the transport layer to the world of Web Services, using the WS-* framework. We use SOAP instead of TCP for handshake messages transfer. WS-Trust and WS-SecureConversation provide the framework

to describe handshake protocol messages, and WS-Security allows us to put security related metadata into the <Security> header element. Our specification of SSL-over-SOAP complies with the recent version of SSL, namely TLS 1.1 [6].

For the handshake protocol, we put the handshake messages into payloads of the SOAP exchange, such that all elements are contained in the body part of the SOAP message—never in the header. This is important to allow us to duplicate the renegotiation feature of the original SSL protocol: An SSL handshake can also be carried out over a connection that is already protected using SSL/TLS (for example, to transfer client certificates in encrypted form). In the case of SSL-over-SOAP, this means that we want to be able to apply WS-Security and WS-SecureConversation even to those SOAP messages constituting a new handshake. In such a situation, the cryptographic parameters used in the WS-Security header stem from the original session, while the parameters of the next session are negotiated using the handshake protocol.

Additionally, we define how the `Finished` message is generated. In the SSL/TLS standards, this is done by computing a pseudorandom function taking as input the exchanged secret and a concatenation of all message bytes exchanged for the current handshake, and then sending this message over the new cryptographically secured channel. Such authentication is important to thwart man-in-the-middle attacks: If an attacker has modified some handshake message to influence parameter negotiation, then verifying the peer's `Finished` messages will reveal that something is wrong (assuming that only ciphersuites providing reasonably strong authentication are ever negotiated).

In SSL-over-SOAP, we use the same pseudorandom function for the `Finished` message. Its inputs are the new master secret and the concatenation of the (serialized canonicalized) bodies of the SOAP messages for the handshake. Putting parts of the handshake messages into the SOAP header instead of the body might make the protocol vulnerable to attacks, or could lead to a much more complex computation of the `Finished` message (see Section 5 for more discussions).

3.2. ClientHello (Message 1)

For convenience, we will refer to the initiating service as "client" and to the responding service as "server", thus adopting SSL terminology. In SSL-over-SOAP, the initial message is the `ClientHello` (Fig. 1). Using the framework of WS-Trust, we embed the messages into the <wst:KeyExchangeToken> within the <wst:RequestSecurityToken> element. The <wst:RequestKET> indicates that an additional message from the server is required to complete the key establishment. The <wst:RequestSecurityToken> has an attribute @tls:Id with a UUID value so that we can reference to this message later.

The <tls:ClientHello> specifies the SSL version of the client (<tls:Version>), the ciphersuites (<tls:CipherSuites>) and compression methods (<tls:CompressionMethods>) as well as the client's nonce (<wst:Entropy>). The version number 3.2 indicates that TLS version 1.1 is used [6, Section 6.2.1]. While the ciphersuite in SSL is identified by two bytes, it is identified here by a URI.

For example, the ciphersuite TLS_RSA_WITH_AES_256_CBC_SHA-1 is identified by http://www.example.org/tls#tls_RSA_with_AES256_SHA-1.

```
<soap:Envelope>
 <soap:Body>
  <wst:RequestSecurityToken tls:Id='uuid:UUID-msg1'>
   <wst:TokenType>.../sc/sct</>
   <wst:RequestType>.../trust/KET</>
   <wst:RequestKET/>
   <wst:KeyExchangeToken>
    <tls:ClientHello>
     <tls:Version>3.2</>
     <tls:CipherSuites>
      <tls:CipherSuite>...#tls_RSA_with_AES256_SHA1</>
     <tls:CompressionMethods>
      <tls:CompressionMethod>...#compression_null</>
     <wst:Entropy>
      <wst:BinarySecret Type='.../Nonce'>M7o9...MO0o=</>
```

FIGURE 1. `ClientHello` message sent from Client to Server in order to initiate the handshake

3.3. ServerHello, Certificate, ServerKeyExchange, CertificateRequest (Message 2)

The second message is the server's response to the security token request (Fig. 2). The response contains mandatory and optional elements. As with SSL, the choice of elements depends on the ciphersuite selected (<tls:ServerKeyExchange>) and on whether the server requests client authentication (<tls:CertRequest>).

<tls:ServerHello> contains the SSL protocol version of the server (<tls:Version>), the session ID (<tls:SessionID>), the ciphersuite selected by the server (<tls:CipherSuite>), the compression method (<tls:CompressionMethod>), and the server nonce (<wst:Entropy>). The session ID is adopted from SSL to manage the session and used to execute the abbreviated handshake.

Fig. 2 illustrates an example message where client and server are mutually authenticated on the basis of X.509v3 Certificates. Such certificates are handled by including a <wsse:BinarySecurityToken> into the <tls: Certificates > element. The type X509PKIPathv1 indicates that this token specifies a certificate chain.

<tls:ServerKeyExchange> would contain key material for DH key exchange. In Fig. 2, client and server opt for key transport based on RSA where the server's public key is provided by the certificate. Hence, <tls:ServerKeyExchange> is an empty tag, and could be omitted.

The <tls:CertificateRequest> element is used to signal the client that is has to authenticate using a X.509 security token. The tag contains security policies

which specify the client certificate's requirements. In Fig. 2, the server requires client authentication and opts for a client certificate that is issued for RSA signatures from the Certificate Authority Test Root CA.

The SSL/TLS message `ServerHelloDone` serves only as a delimiter. We omit `ServerHelloDone` in SSL-over-SOAP, since we combine multiple SSL elements into one <wst:KeyExchangeToken>.

```
1 <soap:Envelope>
2  <soap:Body>
3   <wst:RequestSecurityTokenResponse  tls:Id='uuid:UUID-msg2'>
4    <wst:TokenType>.../trust/KET</>
5    <wst:RequestedSecurityToken>
6     <wst:RequestKET/>
7     <wst:KeyExchangeToken>
8      <tls:ServerHello>
9       <tls:Version>3.2</>
10      <tls:SessionID>Vz2e...4WU=</>
11      <tls:CipherSuite>...#tls_RSA_with_AES256_SHA1</>
12      <tls:CompressionMethod>...#compression_null</>
13      <wst:Entropy>
14       <wst:BinarySecret  Type='.../Nonce'>ihsK...7CYA=</>
15     <tls:ServerKeyExchange>
16     <tls:Certificates>
17      <wsse:BinarySecurityToken
18         ValueType='...#X509PKIPathv1'>MIIC...iw==</>
19     <tls:CertRequest>
20      <tls:Certtype>...#rsa_sign</>
21      <tls:CAs>
22       <tls:CA>CN=Test  Root  CA</>
```

FIGURE 2. `ClientHello, ServerKeyExchange, Certificate, and CertificateRequest` message sent from Server to Client. Server and Client agree on the ciphersuite "#tls_RSA_with_AES256_SHA1".

3.4. ClientKeyExchange, Certificate, CertificateVerify, Finished (Message 3)

Here the message structure becomes a little more complicated, since we have to combine unprotected parts (`ClientKeyExchange`, `Certificate`, `CertificateVerify`) and protected parts (`Finished`) into one SOAP message in order to comply with the SSL protocol specification (Fig. 3).

The <tls:PreMasterSecret> (lines 34–37) within <tls:ClientKeyExchange> (lines 33–37) contains the encrypted premaster secret (recall that client and server negotiated the RSA ciphersuite). The premaster secret is encrypted with the server's public key. Since client authentication has been requested in the previous message, the client makes use of the <tls: Certificates > element (lines

38–40) to send a certificate chain containing its certificate and the certification authority's certificates. The <tls: CertificateVerify > (lines 41–50) has only one child element, <ds:Signature> (lines 42–50). For more details, see Section 4. The <ds:Reference> elements in lines 46–47 reference the exchanged messages (messages 1 and 2) via the @URI with the prefix *urn:uuid*. The last two <ds:Reference> elements in lines 48–48 reference the <tls:ClientKeyExchange> (lines 33–37) and the <tls: Certificates > (lines 38–40) in the same message. The <wst:SecurityContextToken> (lines 28–29) specifies that the master secret should be addressed by the UUID specified, *UUID-sct*.

After choosing a premaster secret, the client computes the master secret and derives the session keys. In order to confirm the correct generation of session keys, it computes the content `Finished` message as follows:

$$client\ finished = \mathtt{PRF}_{master\ secret}\big(\text{“}\mathtt{client\ finished}\text{”}, \\ MD5(exchanged\ messages) \,||\, SHA1(exchanged\ messages)\big)[0...11]$$

In the SSL framework, the exchanged messages are those visible at the handshake layer and do not include record layer headers. Hence in SSL-over-SOAP, the exchanged messages are the SOAP bodies in messages 1 and 2, and the SOAP body except the <tls:Finished> in this message. The SOAP bodies are first canonicalized with the algorithm Exclusive C14N and then concatenated. The result is then used as the input of the hash algorithms MD5 and SHA-1.

The client then constructs the `Finished` message by encoding the result of the TLS PRF: <tls:Finished>Base64(client finished)</>. The <tls:Finished> is then encrypted and signed in the following way:

The client inserts a <wsse:Security> (lines 3–23) into the SOAP header <soap:Header>. Then, the client adds a <wst:SecurityContextToken> (lines 4–5) to <wsse:Security>. This token has the same properties as the token within the SOAP body (lines 28–29). However, we may not move this security token to the body, since only if it is located in the SOAP header, the token can be used for decryption and signature verification. Then, the client computes a derived key token <wsc:DerivedKeyToken> that specifies the client_write_key, and is used to encrypt the content of <tls:Finished> (lines 51–56).The <wsc:DerivedKeyToken> (lines 17–21) and a new <xenc:ReferenceList> (lines 22–23) that locates the encrypted <tls:Finished> are then added to the <wsse:Security> element. Finally, the client computes a derived key token <wsc:DerivedKeyToken> (lines 6–10) that specifies the client_write_MAC_secret used to sign the encrypted <tls:Finished> with the client_MAC_secret. The signature is represented by a <ds:Signature> (lines 11–16). Both elements are then added to the <wsse:Security>.

Note that in message 3 and in message 4 (see Sect. 3.5), the session ID is added to the <wst:KeyExchangeToken> so that the receivers (the server in message 3 and the client in message 4) can chain the messages.

```
<soap:Envelope>
 <soap:Header>
  <wsse:Security>
   <wsc:SecurityContextToken wsu:Id='Id-sct'>
    <wsc:Identifier>uuid:UUID-sct</>
   <wsc:DerivedKeyToken wsc:Algorithm='...#TLS_PRF'
       wsu:Id='Id-clientMACKey'>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI='#Id-sct'/>
    <wsc:Length>20</><wsc:Offset>0</>
   <ds:Signature>
    <ds:SignedInfo>
     <ds:Reference URI='#Id-finished'>
      <ds:KeyInfo>
       <wsse:SecurityTokenReference>
        <wsse:Reference URI='#Id-clientMACKey'/>
   <wsc:DerivedKeyToken wsc:Algorithm='...#TLS_PRF'
       wsu:Id='Id-clientWrtKey'>
    <wsse:SecurityTokenReference>
     <wsse:Reference URI='#Id-sct'/>
    <wsc:Length>32</><wsc:Offset>40</>
   <xenc:ReferenceList>
    <xenc:DataReference URI='#Id-EncFinished'/>
 <soap:Body>
  <wst:RequestSecurityTokenResponse>
   <wst:TokenType>.../trust/KET</>
   <wst:RequestedSecurityToken>
    <wsc:SecurityContextToken>
     <wsc:Identifier>uuid:UUID-sct</>
    <wst:KeyExchangeToken>
     <tls:SessionID>Vz2e...4WU=</>
     <wst:RequestKET/>
     <tls:ClientKeyExchange Id='Id-cke'>
      <tls:PreMasterSecret>
       <xenc:EncryptedKey>
        <xenc:EncryptionMethod Algorithm='...#rsa-1_5'/>
        <ds:KeyInfo/><xenc:CipherData />
     <tls:Certificates Id='Id-certs'>
      <wsse:BinarySecurityToken ValueType=
        '...#X509PKIPathv1'>MIIC...y/Z/</>
     <tls:CertificateVerify>
      <ds:Signature>
       <ds:SignedInfo>
        <ds:CanonicalizationMethod />
        <ds:SignatureMethod Algorithm='...#rsa-sha1'/>
        <ds:Reference URI='urn:uuid:UUID-msg1'/>
        <ds:Reference URI='urn:uuid:UUID-msg2'/>
        <ds:Reference URI='#Id-cke'/>
        <ds:Reference URI='#Id-certs'/>
       <ds:SignatureValue>RR4p...vFvA=</>
     <tls:Finished Id='Id-finished'>
      <xenc:EncryptedData Id='Id-EncFinished'>
       <ds:KeyInfo>
        <wsse:SecurityTokenReference>
         <wsse:Reference URI='#Id-clientWrtKey'/>
       <xenc:CipherData/>
```

FIGURE 3. `ClientKeyExchange`, `Certificate`, `CertificateVerify`, and `Finished` sent from Client to Server.

3.5. Finished (Message 4)

The server's <wst:RequestSecurityTokenResponse> contains the <tls:Finished> message (see Fig. 4). To compute the content of the `Finished` message, the server uses the same `PRF` function as the client except the following differences: 1) the label is "server finished"; 2) for the exchanged message, we refer to the SOAP bodies in messages 1, 2 and 3. In the SSL framework, the headers in the record layer are not considered, hence we first decrypt the message 3 and use the decrypted SOAP body (namely the <tls:Finished> is not encrypted).

The message allows the client to verify that the server has received all the previous messages from the client. As with the previous message, the server's `Finished` has the same SOAP header structure, but differs in the keys used, i.e. the server uses server_write_MAC_secret for the HMAC and server_write_key for the encryption.

```
1 <soap:Envelope>
2  <soap:Header>(Similar as in Message 3)</>
3  <soap:Body>
4   <wst:RequestSecurityTokenResponse>
5    <wst:TokenType>.../trust/KET</>
6    <wst:RequestedSecurityToken>
7     <wst:KeyExchangeToken>
8      <tls:SessionID>Vz2e...4WU=</>
9       <tls:Finished wsu:Id='Id-Finished2'>
10        <xenc:EncryptedData Id='Id-EncFinished2'/>
```

FIGURE 4. `Finished` message sent from Server to Client that confirms the negotiated ciphersuite and derived session keys.

4. A Token-based Protocol

We present additional details for a concrete Web Services protocol that instantiates the framework described in the previous section. Specifically, we show a protocol variant using X.509 certificates as security tokens. The framework can similarly be instantiated using other security token types, requiring other protocol variants: for example, password-based authenticated key exchange using a scheme such as "SOKE" [4] would use user name tokens. Recall that during the execution of the protocol, the services endpoints may decide which authentication token to use.

The X.509v3 binary token authentication protocol is illustrated in Fig. 5. The protocol's goal is to negotiate a tuple of session keys between two services $\mathcal{W}_1$ and $\mathcal{W}_1$, while the services authenticate on the basis of X.509v3 binary tokens. Assuming that in a setup stage the tokens have been stored in credential stores. We denote the certified public pairs of $\mathcal{W}_1$ and $\mathcal{W}_2$ by (pk_1, sk_1) and (pk_2, sk_2), respectively.

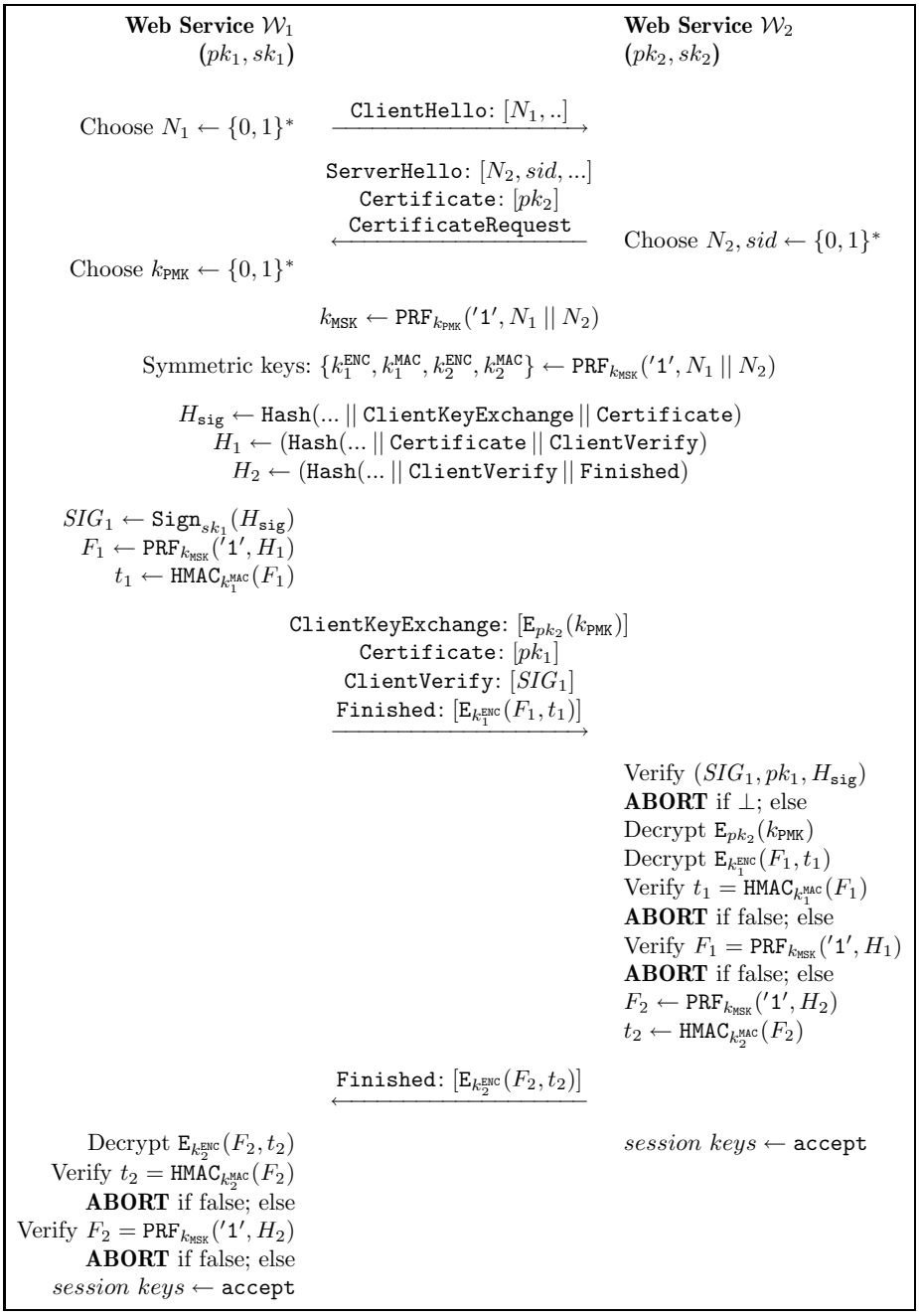


FIGURE 5. Key establishment protocol based on X.509v3 tokens

$\mathcal{W}_1$ initiates the handshake. It randomly chooses a client nonce N_1 and forwards this parameter to $\mathcal{W}_2$. Then, $\mathcal{W}_1$ chooses a nonce N_2 and appends to the nonce its certified public key pk_2. Upon receiving the message, $\mathcal{W}_1$ randomly chooses a premaster secret $k_{\texttt{PMK}}$ and encrypts the premaster secret with the public key pk_2 from $\mathcal{W}_2$. The premaster secret $k_{\texttt{PMK}}$ is used to derive the master secret $k_{\texttt{MSK}}$, using the pseudo-random function PRF parameterized by the services' nonces and the labeling string "`master secret`", abbreviated with "1" in the protocol description. This is the pseudorandom function as specified by SSL. (Other PRF constructions could be used.)

$\mathcal{W}_1$ applies the master secret $k_{\texttt{MSK}}$ to compute the tuple of session keys $\{k_1^{\texttt{ENC}}, k_1^{\texttt{MAC}}, k_2^{\texttt{ENC}}, k_2^{\texttt{MAC}}\}$. Here $k_i^{\texttt{ENC}}$ and $k_i^{\texttt{MAC}}$ are encryption and authentication keys, respectively. $\mathcal{W}_1$ feeds the pseudorandom function PRF with the labeling string "key expansion" and the concatenation of the services' nonces N_1 and N_2. In addition, $\mathcal{W}_1$ proves its identity by signing the digest of previously negotiated messages SIG_1 using its certified private key sk_1. Finally, $\mathcal{W}_1$ confirms the session key generation, using the pseudorandom function PRF that takes as input the master secret $k_{\texttt{MSK}}$, the string "`client finish`" and the hash value H_1 of all previous messages. It then authenticates and encrypts the output $\texttt{F}_1$ with the session keys $\{k_1^{\texttt{ENC}}, k_1^{\texttt{MAC}}\}$.

Upon receiving the message, $\mathcal{W}_1$ decrypts the premaster secret and verifies $\texttt{SIG}_1$, using the client's certified public key pk_1. It ensures that it is connected to $\mathcal{W}_1$, i.e. it checks that $\mathcal{W}_1$ is a valid endpoint. Otherwise $\mathcal{W}_2$ aborts the session. In the positive case, $\mathcal{W}_2$ derives in analogy to $\mathcal{W}_1$ the same master secret $k_{\texttt{MSK}}$ and the same tuple of session keys $\{k_1^{\texttt{ENC}}, k_1^{\texttt{MAC}}, k_2^{\texttt{ENC}}, k_2^{\texttt{MAC}}\}$. Then, $\mathcal{W}_2$ decrypts $\texttt{E}_{k_1^{\texttt{ENC}}}(F_1, t_1)$ and verifies that tag $t_1 := \texttt{HMAC}_{k_1^{\texttt{MAC}}}(F_1)$, where F_1 is the hash over all previous messages. If this verification fails, $\mathcal{W}_2$ aborts the protocol. Otherwise, $\mathcal{W}_2$ confirms the negotiated session keys using the pseudorandom function PRF that takes as input the master key $k_{\texttt{MSK}}$, the labeling string "server finish", and the hash value over all previous messages H_2. It then authenticates and encrypts the output F_2 with the session keys $\{k_2^{\texttt{ENC}}, k_2^{\texttt{MAC}}\}$.

Finally, when $\mathcal{W}_1$ receives the message $\texttt{E}_{k_2^{\texttt{ENC}}}(F_2, t_2)$, it decrypts the message and verifies that tag $t_2 := \texttt{HMAC}_{k_2^{\texttt{MAC}}}(F_2)$, where F_2 is the hash value of all previously received messages. If the verification fails, $\mathcal{W}_1$ aborts the session. Otherwise, $\mathcal{W}_1$ and $\mathcal{W}_2$ start to use the negotiated keys $\{k_1^{\texttt{ENC}}, k_1^{\texttt{MAC}}, k_2^{\texttt{ENC}}, k_2^{\texttt{MAC}}\}$ for symmetric cryptography.

5. Security Discussion

Although our SSL-over-SOAP protocol on the outside looks very different from standard transport-layer SSL/TLS, the handshake quite closely follows the original protocol. We have replaced the SSL/TLS data formats using an XML-based approach, but without changing the cryptographic essence. Thus, previous analyses of the SSL/TLS handshake as appearing in [13–15] apply similarly: the long

experience with SSL/TLS provides evidence that our proposal is cryptographically sound as well.

To get a feel for the cryptographic approach in these protocols (both the original SSL/TLS and our SSL-over-SOAP), observe that most of the handshake negotiation is not cryptographically authenticated immediately. Besides signatures in certificates, authentication appears only in the form of digital signatures if either a `ServerKeyExchange` message is used (the server signs its key share along with the client and server random nonces, thus binding the key share to the current handshake), or if a `CertificateVerify` message is used (in which the client presents a signature on the handshake so far to authenticate itself to the server).

Many typical scenarios involve neither message. An attacker can manipulate the handshake protocol messages being exchanged to influence the handshake outcome: For example, if the client offers multiple ciphersuites in the `ClientHello` message, an attacker could remove the client's preferred ciphersuites from the list, leaving the server with fewer ciphersuites to choose from—such as just those ciphersuites that are the easiest to break. This changes only in the moment when the `Finished` messages are exchanged. These messages cover the complete handshake as well as the resulting master secret, thus retroactively authenticating everything in the current handshake, provided that the master secret could only be known to the legitimate protocol participants. (For example, in an RSA-based handshake, the client encrypts the premaster secret for the server's certified public key, thus ensuring that the premaster secret and thus the master secret remains secret from any attacker.) Accordingly, it is a fundamental security requirement the any party engaging in a handshake only be willing to negotiate ciphersuites that can be assumed to provide security in this sense. Any further security properties, notably those of application data encryption, rely on this.

The `Finished` message is the first piece of data to be encrypted and authenticated under the newly negotiated keys and algorithms, thus also providing a verification that negotiation succeeded as intended and that both parties now are indeed using compatible cryptography. Once the `Finished` messages have been verified, application data is encrypted and authenticated the same way. In the standard SSL/TLS protocol, symmetric authentication is added to the plaintext before encryption.

This is done differently in our SSL-over-SOAP setting (see Fig. 3), where symmetric authentication (following the XML Digital Signatures specification) is applied to the ciphertext. This change is not cryptographically trivial, but does not harm the protocol. The combination of symmetric authentication with encryption can be considered *authenticated encryption* [5]. As discussed in [5], for general composition of an encryption scheme with a MAC, the "encrypt-then-MAC" approach does the best job of providing authenticated encryption. ("MAC-then-encrypt" as used in standard SSL/TLS in general has some problems, although these do not apply to the standard ciphersuites [11].) That is, while SSL-over-SOAP differs from standard SSL/TLS in its use of symmetric cryptography, the approach used in SSL-over-SOAP is in fact cryptographically sound.

6. Conclusion and Outlook

The SSL-over-SOAP approach provides a practical framework for key establishment for Web Services. We use the experience with the practically proven SSL/TLS protocol family for this purpose. This allows us to transfer SSL/TLS protocol ideas to reuse them for Web Services, while giving us much more flexibility and security than direct use of SSL/TLS at the transport layer. Our prototype implementation has shown the feasibility of implementing complex cryptographic protocols within the WS-* framework.

In this paper, we only looked at one basic form of an SSL handshake as an example—an RSA-based handshake (involving an encrypted premaster secret). The SSL-over-SOAP approach applies to many more protocol variants. For example, we can directly transfer the work that has been done in [4] for password-based authenticated key exchange in TLS, where parties rely on low-entropy secrets instead of certificates for authentication. So besides X.509v3 binary token authentication as described in Section 4, we can also specify password token authentication using the "SOKE" scheme from [4]. We plan to complete an open source software library for SSL-over-SOAP, which will offer ciphersuites for both for X.509v3 binary token authentication and for password token authentication.

Our experiences with SSL-over-SOAP should be considered as a starting point for the definition of other key agreement protocols, e.g., the IPsec OAKLEY protocol, or group key agreement protocols. However, security analyses of such protocols can not be directly transferred to the web services world, e.g. considering XML wrapping attacks. Necessary conditions for key agreement protocols to be secure in an XML context (e.g. explicit key confirmation) have to be researched.

References

[1] Security in a Web Services World: A Proposed Architecture and Roadmap, April 7, 2002. `http://www.ibm.com/developerworks/library/specification/ws-secmap/`.

[2] Web Services Secure Conversation Language Specification (WS-SecureConversation), February 1, 2005. `ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation.pdf`.

[3] Web Services Security: SOAP Message Security 1.1 (WS-Security 2004) Working Draft, November 7, 2005. `http://www.oasis-open.org/committees/download.php/15251/oasis-wss-soap-message-security-1.1.pdf`.

[4] M. Abdalla, E. Bresson, O. Chevassut, B. Möller, and D. Pointcheval. Provably secure password-based authentication in TLS. In S. Shieh and S. Jajodia, editors, *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security (ASIACCS'06)*, pages 35–45, 2006.

[5] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976, pages 531–545, 2000.

[6] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) protocol, version 1.1. RFC 4346. http://www.ietf.org/rfc/rfc4346.txt, 2006.

[7] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[8] L. Fang, S. Meder, O. Chevassut, and F. Siebenlist. Secure password-based authenticated key exchange for web services. In *SWS '04: Proceedings of the 2004 workshop on Secure web service*, pages 9–15, New York, NY, USA, 2004. ACM Press.

[9] S. Hada and H. Maruyama. Session authentication protocol for web services. In *SAINT-W '02: Proceedings of the 2002 Symposium on Applications and the Internet (SAINT) Workshops*, page 158, Washington, DC, USA, 2002. IEEE Computer Society.

[10] A. Herzberg. Secure XML transport protocol. Lecture Notes, Chapter 14, 2000. http://www.cs.biu.ac.il/~herzbea/Chapters/Chapter%2014%20XML%20Security.pdf.

[11] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139, pages 310–331, 2000.

[12] M. McIntosh and P. Austel. Xml signature element wrapping attacks and countermeasures. In *ACM Workshop on Secure Web Services*, 2005.

[13] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *7th USENIX Security Symposium*, 1998.

[14] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Computer and System Security*, (3):332–351, 1999.

[15] B. Schneier and D. Wagner. Analysis of the SSL 3.0 protocol. In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, 1996.

[16] W3C Consortium. XML-encryption syntax and processing, 2002. http://www.w3.org/TR/xmlenc-core.

[17] W3C Consortium. XML-signature syntax and processing, 2002. http://www.w3.org/TR/xmldsig-core.

[18] D. Zhang and J. Xu. Multi-party authentication for web services: Protocols, implementation and evaluation. In *Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'04)*, pages 227–234, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

Sebastian Gajek
Horst Görtz Institute for IT Security
Ruhr-Universität
44780 Bochum, Germany
e-mail: sebastian.gajek@nds.rub.de

Lijun Liao
Horst Görtz Institute for IT Security
Ruhr-Universität
44780 Bochum, Germany
e-mail: lijun.liao@nds.rub.de

Bodo Möller
Horst Görtz Institute for IT Security
Ruhr-Universität
44780 Bochum, Germany
e-mail: bmoeller@crypto.rub.de

Jörg Schwenk
Horst Görtz Institute for IT Security
Ruhr-Universität
44780 Bochum, Germany
e-mail: joerg.schwenk@nds.rub.de

Whitestein Series in Software Agent Technologies, 159–170

A Framework for QoS-based Resource Brokering in Grid Computing

Nadia Ranaldo and Eugenio Zimeo

Abstract. Effective and efficient exploitation of Grid computing facilities requires advanced resource management systems to automatically and transparently ensure the fulfillment not only of functional requirements but also of non-functional ones. This paper presents a framework for brokering of Grid resources, virtualized through Web Services, which can be dynamically configured with respect to multiple syntactic and semantic description languages and related matching strategies. Hence, it discovers and selects resources and automatically allocates application tasks to them on the basis of both functional and quality of service (QoS) requirements. In particular, the paper presents a framework specialization which aims to select a pool of resources whose overall performance allows for satisfying time and cost constraints for the execution of an application partitioned in concurrent tasks according to the data parallelism pattern.

Keywords. Resource Brokering, Grid Computing, Quality of Service.

1. Introduction

Thanks to the increasing amount of resources available across the Internet and to improvements of wide-area network performance, in recent years grid computing is emerging as a viable paradigm to satisfy the continuous growth of computation power demand, which often can not be fulfilled exploiting the inner resources of a single organization. This trend is also promoting new business models for providers that would deliver Grid computing functionalities, eventually customized on demand, to host applications and to meet customer needs [1]. After a first generation of solutions based on dedicated technologies, the diffusion of the Web has proposed new architectural (Service-Oriented Architecture - SOA) and technological (Web Services) approaches to address heterogeneity, distribution, security and interoperability in large-scale systems. So, grid applications can be built as the composition of independent services delivered by distributed providers [2]. In particular, a Grid

workflow can be obtained composing domain dependent services, which virtualize the access to specific and high-level utilities delivered by providers that leverage high-performance dedicated clusters and software libraries for complex computations. In this case, functional matching strategies have to be adopted to discover and select the services that deliver the required domain-dependent functionalities for the enactment of the overall application.

More often, differently from B2B environments, Grid workflows orchestrate services that virtualize the access to resources delivering low-level functionalities (called *Grid services*), such as data acquisition, computation and storage. Such services are characterized by heterogeneous assets and performance (due to heterogeneity and sharing of resources onto which they are deployed). As a consequence quality of service (QoS) constraints assume a key role for scheduling resources in the Grid since often the fulfillment of a computational goal is strongly tied to desired performances (execution time, quality of results, reliability, throughput, trust, etc.) and economic costs. For this reason, a lot of research has been devoted to the definition of infrastructure components able to hide heterogeneity (*computing power transparency*) and to satisfy QoS constraints. In a connection model based on middle agents, such as the model proposed by SOA, a fundamental role for achieving the desired transparency is played by resource managers, matchmakers and brokers [3] that should be able to automatically discover available functionalities, choose and schedule the ones that satisfy the QoS constraints specified by the user.

Many compute and data-intensive functionalities in grid workflows (such as linear algebra, image processing, database searching, etc.) are characterized by coarse-grained parallelism that allows for increasing performance by exploiting a pool of distributed resources using parallel computing patterns such as simple parallelism, data parallelism and pipeline patterns [2]. A full exploitation of multiple resources to execute Grid workflows will be reached if the following main issues will be taken into account: (1) the adoption of scheduling techniques based on multi-constrained matching strategies able to find a pool of resources satisfying global constraints; (2) definition of formal languages for QoS description of Grid services in order to avoid ambiguity during matching.

In this paper, we answer to these issues by proposing a framework for QoS brokering of resources, virtualized through Web Services, and its customization. The framework is able to automatically allocate application tasks based on the data parallelism pattern adopting a time and cost-based matching strategy, called *time minimization matching strategy*. We prove, moreover, through an experimental analysis, the validity and accuracy of the system to search and select resources that ensure execution times of real complex applications within prefixed constraints. The resource broker is based on a service matchmaking framework [4] that is extensible and customizable with respect to application scenario through dynamic configuration of syntactic-, structural- and semantic-based discovery and matching strategies, features that make it a suitable and easy-to-use environment to test new scheduling strategies.

The rest of the paper is organized as follows. Section 2 presents related work and technology. Section 3 describes the matchmaking framework. Section 4 illustrates the matchmaker customization for the integration of the time minimization matching strategy. Section 5 presents an experimental analysis of the matchmaking framework. Finally Section 6 concludes the paper and introduces future work.

2. Related Work and Technology

Many Grid systems adopt system-oriented or application-oriented matching heuristics that try to optimize respectively resource utilization and time execution with respect to available resources [5] [6]. G-QoSM [7], a framework for QoS-based service management, focuses on discovery of grid computing services on the basis of QoS criteria and on mechanisms to guarantee QoS levels by means of "contract-based agreements" between service provider and service requester. However, G-QoSM is based on a non standard extension of UDDI, which supports syntactic QoS specifications included in non standardized WSDL-based descriptions of computational services. In our work, instead, we aim to improve the matching process adopting a standardizable and semantic-based description of QoS properties that allow for well formalizing QoS knowledge and so for overcoming syntactic languages limitations due to heterogeneity.

On the other hand, while standard technologies for Web services (SOAP, WSDL and UDDI) define precisely syntax and data structure-based descriptions, there are currently no standards for semantic description and query. A promising solution is the one used in Semantic Web [8]. While for the specification of functional requirements some proposals for standardization of ontologies have been promoted, such as OWL-S [9] and WSMO [10], only recently some results have been reached for QoS requirements. In particular, preliminary efforts can be found in DAML-QoS Ontology [11], QoSOnt ontology [12] and more recently in WSMO [13]. However, such approaches do not take into account QoS aspects that are specific for Grid services and scientific workflows. In fact, beyond to classical QoS attributes defined for a Web Service in a B2B environment, such as reliability, cost, time response, etc., other specific QoS requirements could be specified for Grid services, such as execution time for computational tasks, real-time capability of data acquisition services, minimum storage capability for storage services, etc. In this paper we adopt and extend to the Grid environment the onQoS ontology proposed in [14] for the description of QoS attributes of Web Services. This ontology tries to overcome limitations and scarce homogeneity of many semantic models currently available for QoS description and uses metrics to describe and control the QoS in the matching phase in a quantitative way.

In the context of workflow management, interesting works on QoS have been proposed. A QoS-aware optimization matching approach for workflows is described in [15]. Another approach proposed in [16] takes into account the static scheduling

of workflows modelled as a pipeline of parameter sweep tasks aiming to a fine-grained time optimization in a heterogeneous environment. On the other hand, our approach differs from them because it better focuses on the definition of a flexible brokering framework and matching strategies for QoS-driven and optimized execution of scientific workflows characterized by data parallel tasks that can concurrently exploit multiple resources.

A QoS brokering system which deals with cost constraints is the Grid Service Broker (GSB) [17], which supports access to both computational and data Grids. GSB can transparently access resources that are exposed by various low-level, Grid middleware solutions, such as Globus Toolkit 4 and Alchemi [18] and published on a custom XML-based Grid Market Directory registry. It supports deadline and budget-constrained matching strategies for parameter sweet applications. Heuristics adopted by GSB dynamically allocate a task at a time considering the current state of resources until the budget is consumed. As a consequence they are not useful for scheduling of generic data parallel tasks because do not deal the execution of all the required tasks within a specified deadline. On the contrary our approach aims to grant task execution within specified deadline and budget. As a consequence, it can be effectively adopted in a real environment in which reservation mechanisms (such as in ICENI [19]) grant the availability of a resource with negotiated QoS.

3. Service Matchmaker Framework

The Service Matchmaker [4] is a key component of an ongoing project [20] that aims at defining and implementing a flexible broker for service composition in SOA-based environments. To ensure a high level of flexibility, the framework is designed according to the component framework approach. Its basic infrastructure is able to automatically manage and trigger well-defined activities for discovery and matching of services (figure 1). The architectural skeleton is the *Matchmaker Core*, whilst the specialization is realized through *hot-spots* that permit to customize framework behaviours. The Matchmaker Core, at start-up, uses configuration files to load components that specialize the framework behaviour. Because of the lack of a unique standard language to describe different aspects of a service, the Service Matchmaker supports multi-criteria discovery and matching strategies that can be adopted on different service descriptions (including functional and non functional aspects), each of which describing a specific aspect of a service and eventually adopting a different language.

The matching process starts with a request containing the description of the desired query service (called *template*) submitted by the user through the *Matchmaking API.* As first step, the *Discovery Engine* uses the search functionality offered by the *Registry* to retrieve information on advertised services. The search space of candidate services (called *targets*), initially reduced by the Discovery Engine, is further filtered by the *Matchmaker Manager* that is based on a Pipe and

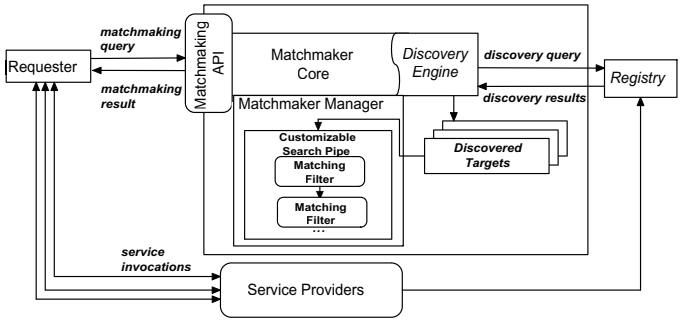


FIGURE 1. Service Matchmaker Framework.

Filter architecture. The Matchmaker Manager adopts two customizable *Search Pipes* of *Matching Filters*, one for functional aspects and one for non-functional aspects. The Search Pipes reduce more and more the service subspace, returning finally a matching result between the template and the targets. The Search Pipes are customized specifying the Matching Filters and their order. A Matching Filter analyzes a specific description of the targets of the received subspace performing a distinct matching strategy and is characterized by a Matching Engine and by a Matching Function (figure 1). The *Matching Engine* receives the subspace of target descriptions and takes care to return the targets that satisfy a matching strategy. Typically, but not necessary, it associates the satisfaction degree of each target with respect to specified requirements for that description invoking the *Matching Function*, which returns a matching score for a specific strategy. The matching of semantic description requires a component with automatic reasoning capabilities. To this end, a *Semantic Matching Function* is configured with respect to an inference engine, through a *Reasoner*, a configurable component whose specialization permits to define the reasoning engine more suitable to specific aims.

4. Service Matchmaker Specialization

The proposed QoS-based resource broker for Grid computing was implemented as a specialization of the Service Matchmaker framework integrating the time minimization matching strategy. This grid-oriented specialization is obtained through an XML-based file that specifies description languages and related functional and non functional matching strategies and through the implementation of the Discovery Engine, Matching Engines, Matching Functions and Reasoners for semantic matching strategies of Grid services that virtualize computational resources.

The family of description languages consists of: (1) WSDL (ver. 1.1), for abstract syntactic description on service interface and concrete syntactic description on bindings and endpoints; (2) OWL-S (ver. 1.1) for abstract descriptions of functional and data semantics; (3) an ontology for QoS description, called GonQoS,

for parameters' description necessary to the time minimization matching strategy. The Discovery Engine specialization interacts with the UDDI registry through the UDDI proxy UDDI4J, [21], an open-source Java implementation of specification for business registry and UDDI API. It performs a minimum functional search exploiting UDDI meta-data on descriptions (for example taxonomy, categoryBag, etc.). In particular, we use a functional aspect-based query which permits a category-based discovery of Grid services through the NAICS taxonomy. The Search Pipe for functional aspects uses three Matching Filters:
- Semantic matching on service operations based on OWL-S language;
- Semantic matching on service input/output and fault based on OWL-S language;
- Structural-syntactic matching on WSDL description of service operations.
The Search Pipe for non functional aspects uses two Matching Filters:
-*Basic QoS-based Matching Filter* (BQMF), for semantic matching on QoS metrics based on GonQoS ontology;
- *Aggregate QoS-based Matching Filter* (AQMF), performing a QoS-based matching strategy that returns the set of services which satisfy QoS requirements in an aggregate manner.

Functional Matching Filters based on ontology descriptions and BQMF are based on the matching approach proposed by Paolucci et al. [22]. The Matching Engine used by such filters is called *One-to-One Matching Engine*. It invokes repetitively the associated Matching Function for each target of the target subspace, assigns to each of them a matching result calculated by the Matching Function, and filters the targets that do not satisfy query criteria. The structural-syntactic Matching Function is based on the strategy proposed by Wang e Stroulia [23]. The Matching Filter AQMF exploits the *One-to-Many Matching Engine*, which processes all the target subspace returned from the previous filter BQMS to satisfy QoS semantic parameters of a query expressed using GonQoS. The result returned by the filter AQMF is based on the time minimization matching strategy, which calculates the portion of the overall workload, specified in the query, to assign to each target. The GonQoS ontology is accessed through a Reasoner which exploits the Jena (ver. 2.4) framework [24] specialized in order to use the inference engine Pellet (ver. 1.3) [25].

4.1. Time Minimization Matching Strategy

The time minimization matching strategy operates with tasks of a Grid application that can be parallelized through the data parallelism pattern [2]: a pool of slaves performs the overall workload, that is decomposed by the master into a high number (but finite) of sub-tasks, called *atomic tasks*, that is the smallest parts of the original workload that can be independently mapped and executed onto different resources without casual precedence relationships.

The application is characterized by the computation size, which corresponds to the total number N of atomic tasks, each of which is characterized by the same complexity in terms of computation, data storage and data transfer aspects. A Grid system is modelled as a finite set $R = \{R_1, R_2, .., R_M\}$ of available resources

communicating through a fully connected wide-area network. Each resource R_i is exposed as a Grid service and is characterized by the following parameters: (1) t_i: the total time for processing an atomic task; (2) c_i: the cost of resource usage for the processing of an atomic task; (3) g_i: the maximum number of atomic tasks (called also capacity) that can be assigned to the resource R_i. Time execution and cost are assumed to be proportional to the amount of atomic tasks assigned to the resource in a linear way. Following the Grid model proposed in [26], the bandwidth on WAN links is not shared and each resource reaches the WAN through a LAN link. In the case of resource reservation mechanisms, only one communication flow goes through the link, that so receives a fixed bandwidth that can be predicted in advance. The QoS parameters specified by the user to model a Grid service request are: (1) the maximum execution time, which represents the deadline, called D, (2) the total available budget, called B, and (3) the capacity of requested data parallel task, corresponding to N.

The time minimization matching strategy regards the problem of finding the "best" set of resources among which to distribute the workload N, so that the aggregate cost for resource usage is lower than budget B (but not necessarily the minimum) and that are able to complete the application execution as quickly as possible (time minimization) and within deadline D. An iterative and low-complexity heuristic to find a near-optimal solution is described in [27].

4.2. GonQoS for Grid Services

The GonQoS ontology for description of QoS parameters of Grid services is based on onQoS [14], an ontology developed using OWL for QoS description, advertising and query of Web services, designed in order to ensure simplicity while maintaining flexibility and extensibility features. It is tied to the OWL-S ontology, which permits to connect a QoS description to the corresponding functional one.

Following the classical approach for ontology definition, GonQoS is organized into three extensible complementary levels. The upper ontology defines the ontological language, which is the basic concepts to model Web service QoS, such as the main properties and restrictions of QoS metrics. In this ontology, a QoS description of a Web Service is represented by a set of QoS metrics. In particular it is necessary to define a new entity of QoSMetric concept for each QoS parameter, that means to define the measured parameter, the measurement scale, the measurement process and one or more measured values belonging to the measurement scale. The middle ontology is a specialization of the first one and is domain independent. Examples are the specialization of QoS parameter of Availability, Performance, Reliability, Cost and Capacity categories. Performance is further specialized in Throughput, Response Time, Latency, etc. The low ontology, that contains domain-dependent specializations of the ontology, defines some grid-specific concepts for QoS definition (see figure 2). The QoS Parameters to characterize query, advertised services and returned result for the time minimization matching strategy are the following.
- *grid-UnitExecutionTime* (seconds): for query: maximum interval time within which a task has to be executed; for advertising: interval time required to execute

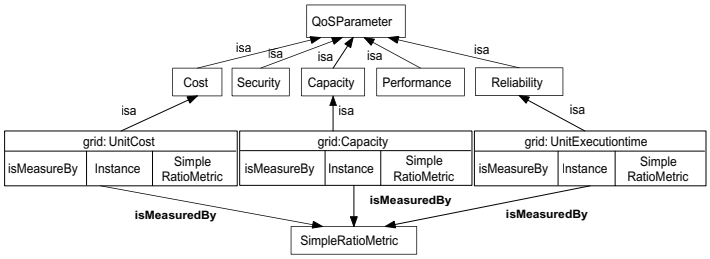


FIGURE 2. GonQoS Ontology.

an atomic task;
- *grid-UnitCost* (euros): for query: maximum budget which can be spent to execute a data parallel task; for advertising: cost for the execution of an atomic task.
- *grid-Capacity* (integer): for query: number of atomic tasks to execute; for advertising: maximum number of atomic tasks which can be executed for a request.

These QoS parameters are classified as Simple Ratio Metric, a specialization of generic QoS metric that permits to define queries adopting relation operators (such as better or equal, tightly less of a certain value, etc.).

5. Matching Strategy Evaluation

The proposed framework for grid resource brokering was tested in order to evaluate its validity and accuracy for the discovery and selection of resources that satisfy deadline and budget constraints through the time minimization matching strategy. An UDDI registry was used for the advertisement of Grid services. For each of them, the providers specify WSDL descriptions, an UDDI categoryBag meta-data for functional aspects and QoS metrics through the GonQoS ontology. The query is formulated through the UDDI categoryBag for functional discovery of computational services and a QoS description of the required deadline, budget and number of atomic tasks to execute. Semantic functional aspects are not exploited in this experimentation, since we consider services virtualizing only computing functionalities. The QoS description is adopted to perform the BQMF in order to throw out the targets which do not satisfy the conditions $c_i \leq B$, $t_i \leq D$, $g_i \leq N$. The AQMF performs the time minimization matching strategy assigning to the filtered targets a part of the query capacity N. A computational service is implemented as a Web Service that takes a certain interval time to execute an atomic task on the basis of performance capability of the resource on which it is deployed. The overall service query is satisfied invoking in a concurrent way the selected services and waiting for their completion.

In this experimentation, we consider ten Grid services deployed in Axis 2.0 container based on Tomcat onto ten distributed resources equipped with a Pentium IV 2.4 GHz and 512 MB of RAM. Resources are inter-connected through a Fast

Service			QoS per Atomic Task	
Template	# instances	Capacity	Execution Time (s)	Cost (euro)
S1	1	100	1.39	60.0
S2	1	100	9.6	10.0
S3	4	100	75.0	4.0
S4	1	100	77.0	4.0
S5	3	100	70.0	5.0

TABLE 1. Testbed Configuration.

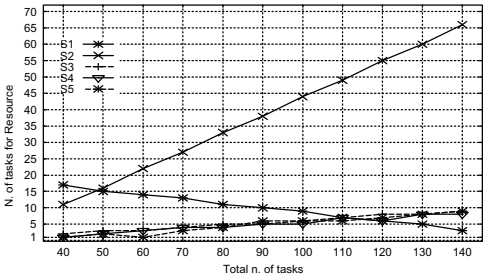


FIGURE 3. Tasks assigned to each service varying N, B=1200.00 euros.

Ethernet LAN that does not cause significant effects on adopted Grid model since data transfers of atomic tasks are kept slight with respect to computational tasks. Resource heterogeneity in terms of cost and performance are emulated taking into account experimentation results previously conducted on a compute-intensive application for power system security analysis [28]. In particular table 1 summarizes the QoS parameters associated to each service. The cost parameters were chosen to be nearly directly proportional to resource performance. In table 1 the service instances with the same QoS parameters are grouped in the same service template. Figure 3 shows the number of atomic tasks assigned to the services considering a deadline of 900.0 s, a budget of 1200.00 euros and a query capacity varying from 40 to 140 atomic tasks. Because of similar capabilities of services with template S3, S4 and S5, the strategy assigns roughly the same number of tasks to each of them. In this scenario the budget of 1200.00 euros is not sufficient to completely exploit expensive services. For this reason the time minimization strategy decreases the number of atomic tasks assigned to the most expensive service, that in this case is the one with template S2, and assigns them to the less expensive ones, until the deadline is not exceeded. Moreover, because the service with template S1 has better performance with respect to services with template S3, S4 and S5, it receives a larger amount of atomic tasks, which increases with the query capacity. Figure 4 (a) shows the estimated execution times considering a query capacity of 180 atomic tasks, deadline of 900.0 s and a varying budget from 1200.00 euros (the minimum value to satisfy the deadline) to 4800.00 euros. The estimated execution time is evaluated as the maximum value among the execution times of each service. We

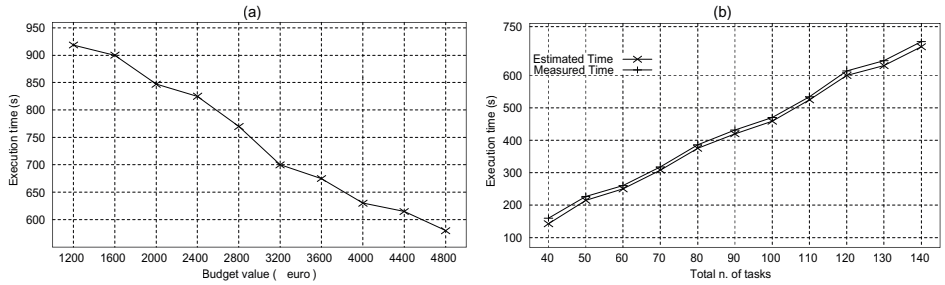


FIGURE 4. (a) Execution time varying budget with D = 900.0 s; (b) Experimental results.

can note that by increasing budget, the algorithm ensures a decreasing execution time thanks to the possibility to allocate more tasks to the expensive and higher performance resources.

Finally, figure 4 (b) shows the execution times that we actually measured running the application on the testbed varying the query capacity. It shows the measured execution times and the estimated execution times by using the time minimization matching strategy. As it is possible to observe, the measured execution times have a nearly linear trend with respect to the atomic tasks to execute, condition that proves the efficiency of the overall system. Such times are, moreover, very near to the execution times estimated by the algorithm. Finally these experimental results proved that the proposed QoS-based brokering framework represents a useful and flexible system for automatically acquiring computational resources when they are necessary, since its accuracy is high and the overhead that users pay for using such system for performing complex tasks is negligible if compared to the improvement of performance and usability.

6. Conclusion

The paper presented the design and evaluation of a framework for QoS brokering of Grid resources virtualized by Web Services. It is based on the Service Matchmaker, a framework that delivers customizable syntactic and semantic discovery and matching strategies. In this work, we presented its customization for supporting the selection of resources among which to distribute the workload of a data parallel task with the aim to minimize execution time and to satisfy deadline and budget constraints. The integration of service invocation mechanisms through workflow technologies, in order to make automatic and transparent to the user the distribution and deployment of applications on multiple resources, will be taken into account in a future work. In particular, we are currently focusing on a dynamic composition and binding technique of services able to transparently and hierarchically distribute applications based on the data parallelism pattern, following the

approach proposed by the authors in [29] for the specification of a partition policy of input data and of an assembling policy of results.

References

[1] I. Foster, C. Kesselman, J. Nick, S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Technical Report, Open Grid Service Infrastructure WG, Global Grid Forum, (2002).

[2] C. Pautasso, G. Alonso, *Parallel Computing Patterns for Grid Workflows*. In: Proceedings of the HPDC Workshop on Workflows in Support of Large-Scale Science (WORKS06). (June 2006).

[3] K. Krauter, R. Buyya, M. Maheswaran, *A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing*. Intern. Journal of Software, Practice and Experience, Wiley Press **32(2)** (2002) 135–164.

[4] G. Tretola, E. Zimeo, *Structure Matching for Enhancing UDDI Query Results*. In: Proceedings of the Int. Conf. on Service Oriented Computing and Applications. (2007).

[5] K. Li, *Job Scheduling and Processor Allocation for Grid Computing on Metacomputers*. Journal of Parallel and Distributed Computing. **65(11)** (2005) 1406–1418.

[6] T. Braun, et al., *A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems*. Journal of Parallel and Distributed Computing *61(6)* (2001) 10–837.

[7] M. Li, P. van Santen, D.W. Walker, O.F. Rana, M.A. Baker. *PortalLab: A Web Services Oriented Toolkit for Semantic Grid Portals*. In: Proceedings of the IEEE CCGrid. (2003) 190–197.

[8] H. Tangmunarunkit, S. Decker, C. Kesselman, *Ontology-based Resource Matching in the Grid - The Grid Meets the Semantic Web*. In: Proceedings of the ISWC, (2003) 706-721.

[9] OWL-S. An OWL-based Web service ontology, http://www.daml.org/services/owl-s

[10] WSMO. http://www.wsmo.org/

[11] C. Zhou, L.T. Chia, B.S. Lee, *DAML-QoS Ontology for Web Services*. In: Proceedings of the International Conference on Web Services. (2004) 472–479.

[12] G. Dobson, R. Lock, *QoSOnt: an Ontology for QoS in Service-Centric Systems*. UK e-Science AHM. (2005).

[13] I. Toma, D. Foxvoug, M.C. Jaeger, D. Roman, T. Strang, D. Fensel, *Modeling QoS Characteristics in WSMO*. In: Proceedings of the Middleware for Service Oriented Computing Workshop (MW4SOC 2006). (2006) 42–47.

[14] E. Giallonardo, E. Zimeo, *More Semantics in QoS Matching*. In: Proceedings of the Int. Conf. on Service Oriented Computing and Applications. (2007) 49–54.

[15] C. Zhang, R.N. Chang, C. Perng, E. So, C. Tang, T. Tao, *QoS-Aware Optimization of Composite-Service Fulfillment Policy*. In: Proceedings of the IEEE SCC. (2007) 11–19.

[16] T. Ma, R. Buyya, *Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids*. In: Proceedings of the IEEE SBAC-PAD. (2005) 251–258.

[17] S. Venugopal, R. Buyya, L. Winton, *A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids*. Technical Report, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia. (2004).

[18] A. Luther, R. Buyya, R. Ranjan, S. Venugopal, *Alchemi: A .NET-Based Enterprise Grid Computing System*. In: Proceedings of the 6th Int. Conf. on Internet Computing. (2005).

[19] A.S. McGouch, A. Afzal, J. Darlington, N. Furmento, A. Mayer, L. Young, *Making the Grid Predictable through Reservation and Performance Modelling*. The Computer Journal, Oxford University Press **48(3)** (2005) 358–368.

[20] LOCOSP project. http://plone.rcost.unisannio.it/locosp

[21] UDDI4J. http://www.uddi4j.org

[22] M. Paolucci, T. Kawmura, T. Payne, K. Sycara, *Importing the Semantic Web in UDDI*. In: Proceedings of the Web Services, E-Business and Semantic Web Workshop (CAiSE 2002). LNCS, Springer-Verlag **2512** (2002) 815-821.

[23] Y. Wang, E. Stroulia, *Flexible Interface Matching for Web-Service Discovery*. In: Proceedings of the IEEE WISE, (2003) 147–156.

[24] Jena 2.4, http://jena.sourceforge.net/

[25] Pellet Reasoner, 1.3, April 17, 2006, http://www.mindswap.org/2003/pellet/

[26] L. Marchal, Y. Yang, H. Casanova, Y. Robert, *A Realistic Network/Application Model for Scheduling Divisible Loads on Large-Scale Platforms*. In: Proceedings of the Int. Parallel and Distributed Processing Symposium. (2005).

[27] N. Ranaldo, E. Zimeo, *An Economy-driven Mapping Heuristic for Hierarchical Master-Slave Applications in Grid Systems*. In: Proceedings of the IEEE IPDPS. (2006).

[28] Q. Morante, N. Ranaldo, A. Vaccaro, E. Zimeo, *Pervasive Grid for Intensive Power System Contingency Analysis*. IEEE Trans. on Industrial Informatics, **2(3)** (2006) 165–175.

[29] N. Ranaldo, E. Zimeo, *A Transparent Framework for Hierarchical Master-Slave Grid Computing*. In: Proceedings of the EuroPar06 - CoreGrid Workshop on Grid Middleware, LNCS, Springer-Verlag **4375** (2007) 74–86.

Acknowledgment

The work described in this paper is framed within the activities of the Core Grid FP6 Network of Excellence funded by the European Commission.

Nadia Ranaldo
Department of Engineering - University of Sannio
82100 - Benevento - Italy
e-mail: `ranaldo@unisannio.it`

Eugenio Zimeo
Department of Engineering - University of Sannio
82100 - Benevento - Italy
e-mail: `zimeo@unisannio.it`

Whitestein Series in Software Agent Technologies, 171–185

Model-Driven Performance Evaluation for Service Engineering

Claus Pahl, Marko Bošković and Wilhelm Hasselbring

Abstract. Service engineering and service-oriented architecture as an integration and platform technology is a recent approach to software systems integration. Software quality aspects such as performance are of central importance for the integration of heterogeneous, distributed service-based systems. Empirical performance evaluation is a process of measuring and calculating performance metrics of the implemented software. We present an approach for the empirical, model-based performance evaluation of services and service compositions in the context of model-driven service engineering. Temporal databases theory is utilised for the empirical performance evaluation of model-driven developed service systems.

Keywords. Service-oriented Architecture, Model-Driven Development, Performance Evaluation, Instrumentation.

1. Introduction

The complexity of software makes its development costly and error-prone. Model-driven engineering (MDE) is an approach to deal with complexity by making software models primary artefacts of the development process. A model is closer to the problem domain than to the underlying implementation. Therefore, it moves the focus of software engineering from technology-specific implementation to the problem domain. MDE utilises two aspects of models. Firstly, complete implementations can be generated from models, but more importantly here, predictions about a software system can be made based on a model. A fully model-based approach hide code-level details and allows a software architect to concentrate on design-stage artefacts.

To provide trustworthy software, quality attributes [4] have to be satisfied. Quality aspects have not been addressed in sufficient depth in the context of heterogeneous, distributed, service-based systems. Service engineering and service-oriented architecture as an integration and platform technology is a recent approach to service-based software system integration. Performance as one of quality attributes, defined as a degree to which a software system meet its objectives for timeliness [5], is of central importance in this context.

At present, research in model-driven performance engineering is mostly dedicated to simulation and performance prediction with mathematical analysis methods [6, 7]. Nevertheless, predictions have to be validated when a software system is implemented and deployed. Validation should be based on modelling constructs as predictions are made according to them. Currently, timing behaviour is analyzed based on source code constructs (e.g., method execution time). In MDE, the level of abstraction is raised. Consequently, observations should be based on modelling constructs, such as components and their states, activities, interactions or methods.

In software engineering, instrumentation is the process of adding software probes to the program [5]. Software probes are additional pieces of code for collecting data about the software execution. A model-based language for instrumentation needs to be derived. Instrumentation languages can enforce data collection in relational manner.

We investigate the empirical performance evaluation of model-driven service-based systems. We focus on composed (or orchestrated) services processes and address their performance behaviour. We present work-in-progress that comprises:

- an instrumentation notation for service models that allows specific service model elements such as services or composition and flow operators to be annotated and marked as providing performance-relevant time information at execution time. We use UML activity diagrams to express service compositions and base our instrumentation language on this UML diagram format. Our work follows others in using UML beyond classical software design. The Erickson-Penker Business Extensions for UML [2] for instance permits UML to document an entire business enterprise.
- model-driven transformation techniques that generate executable code including the monitoring instructions necessary to record time information.
- a trace analysis query language. This language provides the ability to calculate performance metrics such as response time and throughput. The evaluation is based on temporal databases theory [8]. The temporal databases theory relates facts stored in a relational manner with time information. A relational trace is a dynamic list of events and timing information generated by the program as it executes [9]. A query language allows the evaluation based on the traces in terms of service model elements.

Empirical code-level instrumentation and analysis has been investigated in depth. Simulation and analytical models have been used to provide support at design

stages of the development process. Our contribution represents a novel approach for the empirical, model-level evaluation of performance for service-based software systems that

- firstly, an empirical and, thus, ultimately more accurate and reliable technique than simulation and analysis,
- secondly, a fully model-based evaluation technique for the architect that hides code-level details.

The paper is structured as follows. The next section gives an overview of model-driven engineering and service engineering. Motivation and foundations of performance engineering are presented in Section 3. Section 4 introduces the instrumentation language. Performance monitoring through code generation and instrumented execution is described in Section 5. The analysis of evaluation data is discussed in Section 6. Related work is discussed in Section 7 and Section 8 concludes the paper.

2. Model-Driven Development and Service Engineering

The general idea of model-driven engineering (MDE) is to introduce a model as a first-class entity. With models, the development focus is moved to the problem domain. Models often enable the exploitation of formal methods. With abstraction, the understanding of the problem and its realization can be improved. Often, a complete implementation can be generated [10]. Model Driven Architecture (MDA) [11] is one approach for MDE initiated by the Object Management Group (OMG), a consortium of software vendors and users. MDA is based on three ideas: direct representation to shift the focus of software development away from technology toward the problem domain, automation to mechanize the relation of semantic concepts of problem domain and implementation domain, and open standards to enable interoperability to close the semantic gap between domain problems and implementation technologies. Our aim is to enable the evaluation of service performance when the primary artefact is a service (or service process) model.

A service is defined as a piece of software, whose public interfaces are defined and described using an interoperable format. Other systems can interact with the service in a manner prescribed by its definition. The composition of services to orchestrated processes is a major concern in current Web service research [12, 13]. These recent developments have strengthened the importance of architectural questions such as service composition.

Modelling can support these architectural questions. Behaviour and interaction processes are central modelling concerns for service-based software architectures. Fig. 1 illustrates how a UML activity diagram can be used to express a service orchestration – at an abstract level without addressing individual service providers. Four services that provide an online bank account facility login, balance, transfer, and logout are orchestrated into a process starting with a login,

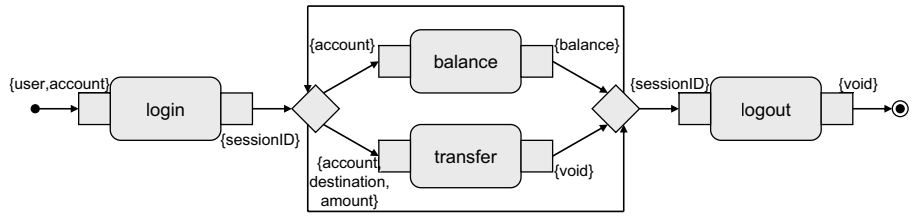


FIGURE 1. Service Process modelled using a UML activity diagram.

then allowing a user to iteratively choose between balance enquiries and money transfers, before logging out.

Explicit models enable developers and clients of services to create reliable service architectures using tool support. A model-driven development approach can even support automated code generation and performance analysis. Assuming that concrete, provided services are already attached to each service element, then an executable WS-BPEL process for the Web service platform can be generated. As we are going to demonstrate, the service composition model can be instrumented for empirical performance analysis and executable processes including performance monitoring functionality can be generated.

3. Performance Evaluation

3.1. Software Performance, Evaluation and Motivation

Performance is considered as the degree to which a software system or component meets its objectives for timeliness [5]. It can be evaluated with simulation, analytical modelling or empirically [9]:

- Simulation is an imitation of a program execution focusing on specific aspects. It is less expensive than building a real system for empirical evaluation. It is flexible as changes can be dealt with easily if the simulation is derived automatically. However, simulation can suffer from a lack of accuracy.
- Analytical modelling is a technique where a system is mathematically described. Results of an analytical model can be less accurate than real-system measurements. However, analytical models are often easy to construct.
- Empirical evaluation is performed by measurements and metrics calculation. They provide the most accurate results as no abstractions are made.

The downside of performance evaluation by implementation, however, includes hardware dependency, extra cost of creating a prototype and deploying it, implementation deficiencies, and challenges in representative workload creation. Two observations led us to consider model-based empirical evaluations. Firstly, an approach for empirical evaluations of software performance for service-based software

systems is still lacking despite its accuracy benefits. Secondly, empirical measurements and evaluations are currently performed only at the code level and mostly based on code constructs.

In model-driven engineering, observations of behaviour should be in terms of modelling constructs. Instrumentation for observing software should also be expressed in terms of these constructs in order to prevent the software architecture from having to represent transformation details and having to deal with code-level details. A necessary part of empirical performance evaluation is the execution data collection through instrumentation.

3.2. Instrumentation

Instruments and instrumentation are commonly used for observing system behaviour and evaluating system properties in a range of disciplines. In software engineering, instrumentation is the process of adding software probes to a program [5]. Software probes are pieces of code for collecting data about the software execution. Two techniques for data collection exist:

- Sampling is a technique where parts of a program are sampled during its execution in some time interval - an example is sampling the program stack to follow program execution. It is a statistical technique in which a representative sample of data about the execution is taken. An advantage is that the impact on the performance of the program does not depend on the execution of the program. However, collected samples are different from run to run. The possibility that infrequent events are missed is another drawback.
- Event tracing is a process of generating traces of events in the software execution. A program trace is a dynamic list of events generated as the program executes [9]. A trace contains time-ordered events and can be used to characterize the overall program behaviour. Problems can be caused due to measurements. Each probe that is added causes execution overhead (performance) and event traces require resources (memory).

Due to its greater reliability, event tracing is used here. Event tracing is also more suitable for service-based software where the focus is on services as black-box entities that interaction in compositions. Traces are presented in our approach in relational manner using the concepts of temporal database theory to support the performance evaluation of traces.

3.3. Temporal Databases

Temporal databases support a notion of time [8]. In contrast to conventional databases, in which only facts are stored, each fact stored in a temporal database is associated with some time information. These facts can be related to a valid time dimension and to a transaction time dimension. The valid time dimension is related to the time when the fact was true in reality. The transaction time dimension is related to the presence of the fact in the database.

Temporal databases which store only facts about the past are called historical databases [8]. Historical databases define two kinds of relations, event and interval

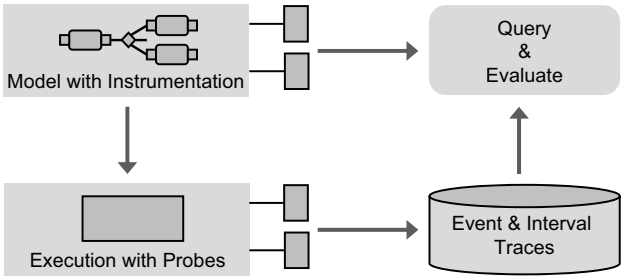


FIGURE 2. Overview of the Framework.

relations [14]. Interval relations are used for storing facts which were true for some time interval. Event relations are used for storing facts which were true at some particular time point.

We utilise concepts from historical databases, such as both interval and event relations, to instrument service composition models.

4. Instrumentation

The execution of a program, such as execution and interactions of a composite service, can be characterized in terms of event and interval relations. For instance, if an element of a modelling language models a part of the program execution which lasts for some time interval, it can be instrumented by a specialization of the interval trace. Our instrumentation technique is developed around an instrumentation language, which is integrated with the service modelling language, i.e. is an extension of the UML activity diagrams that we use to model service orchestrations. Both service orchestration language and instrumentation language are presented at the meta-model layer. We present an overview of the approach in Fig. 2 that relates modelling and execution.

4.1. Service Process Meta-model

The banking example based on the orchestration of services to a service process from Fig. 1 is formulated in terms of a UML activity diagram. A (simplified) definition of UML activity diagrams as a process language is based on activity nodes and edges to represent services and their connectivity, respectively. We have given preference to UML activity diagrams over other process notations such as BPMN, because of UML's elaborate language extension mechanisms.

4.2. Instrumentation Meta-model

Our instrumentation notation comprises of two parts. Firstly, a basic trace package (Fig. 3) to capture the notion of traces, i.e. event and interval traces, and operations

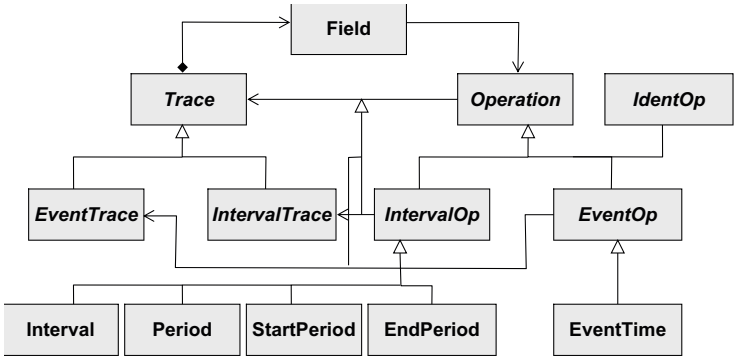


FIGURE 3. Basic Trace Package.

to capture these traces. Secondly, the instrumentation of activity diagrams using the MOF profile extension mechanism (Fig. 4).

The basic trace package reflects the required time dimensions and the recording concepts. The activity diagram instrumentation utilises these. This separation allows the basic instrumentation principles to be reused in a range of problem-specific or even model-specific circumstances – which is important as domain-specific languages are increasingly important. In the given instrumentation, actions as the central elements of activity diagrams and all control nodes are annotated. The execution of actions, which represent services at the model level, takes some time, i.e. an interval trace should be recorded at performance evaluation or execution time. We assume control flow decisions such as start and end of the overall process or choices and mergers as instantaneous events, i.e. modelled as event traces. This is a decision that can be modified at the Instrumentation Diagram level, without affecting the basic trace package. This provides for easy adaptability of the instrumentation to different interpretations and modelling languages.

4.3. Instrumentation Application

The application of the instrumented activity diagram is illustrated in Fig. 5. Two types of model elements - actions such as login or transfer and control nodes such as the start or the first decision point - are instrumented. An interval consisting of begin and end time of the service executions that implement the actions are recorded as a consequence of this instrumentation. Events, i.e. individual time stamps, are recorded for the control nodes.

For the service architect, it is import to find an adequate instrumentation that provides answers to the relevant performance questions. For instance, in a particular situation only the response times (average, maximum) of particular services, such as the account management services balance and transfer, are of interest. Then, the instrumentation needs to reflect these requirements.

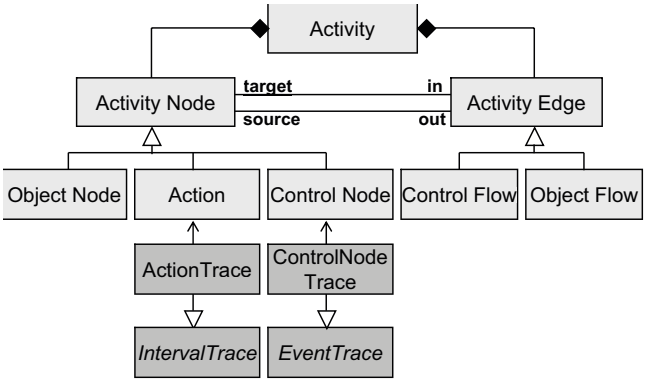


FIGURE 4. Activity Diagram Instrumentation.

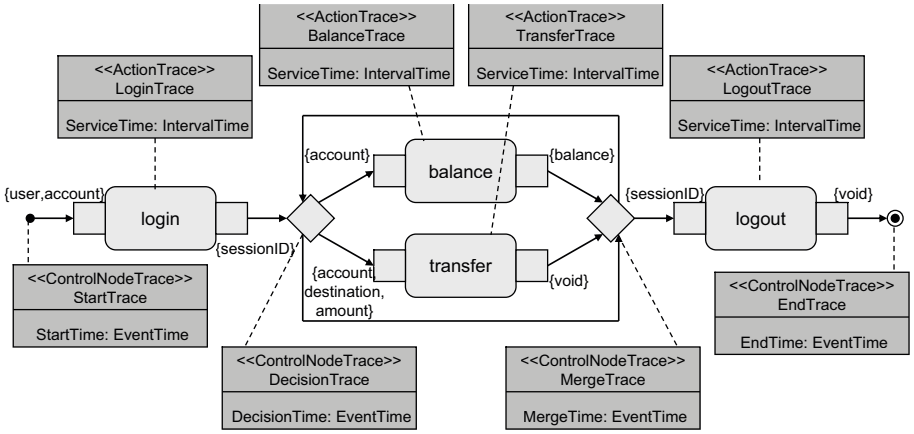


FIGURE 5. Application of the Instrumentation to the online banking service process.

While we consider this instrumentation of actions and control nodes to be the standard, the approach is flexible enough to accommodate context-specific customisations. Some control nodes could be excluded or other modelling elements could be added. This is only limited by the extent to which transformation and code generation support the different model element instrumentations. The instrumentation of elements could be disabled that are difficult to implement or whose analysis would not provide useful performance information.

5. Performance Monitoring

The implementation of the instrumentation should be, firstly, easy to realise and, secondly, implemented without significant overhead. Aspects and interception techniques can be utilised to implement the instrumentation and data collection. Although an important aspect of performance evaluation, the focus of this paper is on the model-related issues of instrumentation specification (such as instrumentation meta-models) and data query and assessment activities (which are discussed later on). We only discuss principles of monitoring here.

5.1. Intrumentation and Monitoring

In the services context, often the problem arises that the addition of probes into the service implementation is not possible due to the nature of services as true black box software components. We therefore distinguish two scenarios:

- controlled environments that allow access to code. Aspect Oriented Programming (AOP) [15] is a programming approach, suitable for the controlled approach, which can be used for transparent software instrumentation. AOP is a technique that enables the separation of instrumentation from the development of the core software functionality [16, 17]. Marenholz et al. [17] use AspectC++ for the instrumentation of operating systems for debugging, profiling/measurement, and runtime surveillance/monitoring.
- open environments in which services are black-box components. For a transparent instrumentation of component systems, interceptors can be used. Interceptors are similar to AOP and can intercept method invocations to transparently instrument a program [18, 19]. Software probes can be predefined and placed in stubs and skeletons during an interface description compilation [19]. The probes can be turned on and off at runtime.

The JBoss Application Server, for instance, enables the transparent aspect-oriented addition of functionality. Its AOP features allow the interception of events and addition of trigger functionality based on those events.

5.2. Generation of Instrumented Code

Aspect Oriented Programming, interceptors, and bytecode and platform instrumentation are approaches that enable the collection of data without affecting the functionality. We utilize these ideas to collect data about the software execution at the model level as a separate concern.

The first step, however, is the generation of executable and instrumented code. Activity diagrams that model service orchestrations can be converted into executable Web services processes, if invocation information such as the service location is added to the abstract service process description:

- AOP concepts are used to generate the instrumented executable service code.
- Interception mechanisms add instrumentation and data collection.

TransferTrace: ServiceTime: IntervalTime	
2:22	2:45
3:03	3:12
3:15	3:29

DecisionTrace: DecisionTime: EventTime
2:19
2:50
3:01
3:10
3:35

FIGURE 6. Collected Data for Online Banking Process Instrumentation.

We propose ATL transformations – the ATLAS Transformation Language ATL is a tool-supported hyrbid model transformation language for the ecplise platform – to transform activity diagrams into AOP-based code.

5.3. Performance Data Recording

The instrumentation includes monitoring and data collection functionality. Data is stored in a historical database such as TimeDB or temporal features in Oracle database servers. Fig. 6 shows a sample recording for the composite process for online banking based on the instrumentation defined in Fig. 5. Here, only data for the decision node and the transfer service are provided as samples.

6. Performance Evaluation

6.1. Analysis Language

Temporal and historical databases – which provide the conceptual background for the analysis part of our evaluation technique – are usually extensions of traditional relational databases. SQL is therefore available as a query language to retrieve information in relation to the recorded event and interval times and to use the language for common statistical operations.

The objective is to extract performance-relevant information from the basic times stored in a historical database that allows a software architect to assess the overall performance of individual services and also orchestrated processes. SQL is sufficient as a query language to formulate the relevant performance assessment queries. More advanced solutions like data warehouses with their extended evaluation support are not required. We can classify performance assessments as follows:

- Response time assessment: response times of activities are usually recorded as intervals. The SQL aggregate functions, such as average AVG or maximum MAX, provide the relevant answers.

- Frequency and distribution of invocations: the distribution of invocations (workload) between the individual services can be determined based on the calculation of ratios between total numbers of invocations.

The database representation directly reflects the modelling layer, as the representation is generated from the model instrumentation. The central goal of fully model-based performance evaluation is therefore achieved. The queries can consequently be formulated in terms of relevant model elements - which is one of the central objectives of model-driven quality engineering.

6.2. Performance Analysis

We have already classified the different types of performance assessments in the previous section. We now illustrate these types.

The average response time for service 'transfer' can be determined as follows:

```
SELECT AVG(ServiceTime)
FROM   TransferTrace
```

The determination of the maximum time can be formulated analogously. In the SOA context, where individual services are often provided by external organisations, this information is usually part of contracts and service-level agreements.

The proportion of 'transfer' invocations based in relation to all user selections (decisions) can also be formulated:

```
SELECT COUNT(ServiceTime) / COUNT(DecisionTime)
FROM   TransferTrace, DecisionTrace
```

This would allow a software architect to judge the frequency of individual service activations in typical application scenarios.

7. Related Work

There are several approaches for analytical evaluations of software performance from annotated models [7, 20] and simulation [21, 26]. A detailed survey related to performance prediction can be found in [6]. There are also several approaches for measurement and instrumentation in the context of code-centric development. Our contribution is an empirical instead of analytic technique for model-level evaluation.

- Snodgrass [22] introduces a relational approach for monitoring systems. His work shows that a relational data structure can be an appropriate formalism for monitoring dynamic behaviour of a system. The programmer manually defines the instrumentation according to concepts of an existing system. Our approach provides a schema for the definition of instrumentation languages according to the modelling formalism used for the specification of programs.
- Liao et al. [23] introduce a high-level language for program instrumentation and monitoring. A programmer specifies monitoring and measuring information, based on which a static analysis of code is done and instrumentation is added. However, their language is suited only for procedural languages.

- Another language for program instrumentation is the Metric Description Language (MDL) [24]. MDL has the ability to define instrumentation as a separate concern, independent of the program functionality, define points at which measurement actions should take place and weave these into a program at runtime. However, it is limited to functionally decomposed systems.

The idea on integrating software models and instrumentation is introduced in [25]. The authors develop a set of tools for a model-driven instrumentation. They define different program models such as a functional program model, a functional implementation model, a performance model and a monitoring model. Based on the monitoring model, the source code is instrumented and trace descriptions are generated. In our approach, the primary artefact of software development is a model. Therefore, instrumentation is done at the model level. The functional implementation model is actually a product of software development. Furthermore, instrumentation defines what to measure and where to measure, and from these two models, automatically source code is produced.

In the specific context of service-based modelling, a lack of performance analysis is even more evident, although the need to address performance is recognised [26] and some solutions exist. The Application Response Measurement (ARM) standard [1] enables the collection of performance data. It is a conceptual library suited for usage with programming constructs. We relate our data collection with modeling constructs on a higher level of instrumentation. The standard only enables the collection of data, but not a systematic way of analyzing. ARM is only an interface for measurement. We introduce a systematic approach for data analysis as well as for instrumentation. A different architectural approach is taken in workflow management contexts [3]. Techniques in available workflow management languages and systems exist that provide timers, which allow to measure the invocation times out-of-the box by the workflow engine itself. Our solution adds a summarization component that performs arithmetic functions over a configurable period of time in a different architectural setting.

8. Conclusions

Empirical performance evaluation enables the validation of timeliness of a software system. In particular in service-oriented architecture, where software quality is paramount, the empirical approach that evaluates concrete application platforms is promising. However, currently an approach for empirical performance evaluation in the service development process where a model is the primary software artefact, is lacking. In order to provide software architects with tools for the reliable evaluation of performance, which goes beyond the predictive approaches of simulation and abstract analysis, a fully model-based instrumentation and analysis technique is necessary. The benefit of an empirical technique over predictive approaches is increased accuracy and therefore reliability of the evaluation results.

We have presented a service-specific approach for the empirical performance evaluation of model-driven developed services. Instrumentation and empirical performance evaluation is at the moment done based on programming language constructs at the source code level. Instrumentation at source code level for data collection about program executions in terms of modelling elements can be error-prone and can require significant effort. Therefore, the instrumentation needs to be done at the model level. The models for software functionality definition and instrumentation definition are separated to reduce the complexity of models.

Our contribution is based on a basic package for the definition of instrumentation languages for UML-based activity diagrams to model service orchestrations, a methodology for deriving instrumentation languages, and a query language for performance metrics calculation. The instrumentation languages enable automatically generated data collection in terms of modelling language constructs, and are stored in the format of relational traces. Temporal database theory provides the background for the monitoring and analysis elements of the evaluation technique.

The instrumentation language is designed to be generic. The basic instrumentation package is application-independent and is, since it is separated from the application-specific instrumentation of specific model elements, transferable to other modelling notations and modelling domains. In order to demonstrate the flexibility of this approach, applications of our framework to class and state models are being investigated.

Currently, our generation and execution platform is not fully implemented. We plan to critically evaluate the feasibility of the approach by integrating software performance evaluation based on relational traces in some commercial tools for model-driven development. Furthermore, experiments on an extensive case study will be performed in order to show what the impact of instrumentation code and execution of the instrumented application is.

Another aspect specific to services shall be investigated in more detail. Our current work neglects specific issues arising from heterogeneous and fully distributed systems. The models we have considered here are activity diagrams that focus on the functional composition of services. The models used do not include the concept of distribution. Activity diagrams, however, allow modelling of distribution through activity partitions (so-called swimlanes). Since especially performance aspects apply to distributed service invocations, corresponding edges that cross partitions could be instrumented and the system could be monitored with respect to these inter-location invocations. We expect spatial-temporal databases to provide the foundations for this aspect.

References

[1] Open Group. *Application Response Measurement - ARM*. 2002.

[2] M. Penker and H.E. Eriksson. *Business Modeling With UML: Business Patterns at Work*. Wiley, 2000.

[3] A. Kumar, W.M.P. Van Der Aalst, and E.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.

[4] W. Hasselbring and R. Reussner. Toward trustworthy software systems. *Computer*, 39(4):91–92, 2006.

[5] C.U. Smith and L.G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, 2001.

[6] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

[7] Object Management Group. UML Profile for Schedulability, Performance, and Time Specification, OMG document formal/05-01-02. web: http://www.omg.org/cgi-bin/apps/doc?formal/05-01-02.pdf, 2005.

[8] C. Zaniolo, S. Ceri, C. Faloutsos, R. Snodgrass, V. S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann Pubishers, 1997.

[9] D.J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000.

[10] B. Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5):19–25, 2003.

[11] Object Management Group. MDA Guide. web: http://www.omg.org/cgi-bin/doc?ormsc/06-06-02.pdf, 2006.

[12] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services – Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[13] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice (2nd Edition)*. SEI Series in Software Engineering. Addison-Wesley, 2003.

[14] N.L. Sarda. Extensions to SQL for Historical Databases. *IEEE Transactions on Knowledge and Data Engineering*, 02(2):220–230, 1990.

[15] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proc. of European Conf. on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, 1997.

[16] M. Debusmann and K. Geihs. Efficient and Transparent Instrumentation of Application Components using an Aspect-oriented Approach. In *14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2003)*, volume 2867 of *LNCS*, pages 209–220. Springer, 2003.

[17] D. Mahrenholz, O. Spinczyk, and W. Schroeder-Preikschat. Program Instrumentation for Debugging and Monitoring with AspectC++. In *Proc. 5th Int. Symp. on Object-Oriented Real-Time Distributed Computing ISORC '02*, pages 249–256. IEEE Computer Society, 2002.

[18] M. Debusmann, M. Schmid, and R. Kroeger. Measuring End-to-End Performance of CORBA Applications using a Generic Instrumentation Approach. In *ISCC '02: Proc. 7th Int. Symp. on Computers and Communications*, pages 181–186. IEEE Computer Society, 2002.

[19] J. Li. Monitoring of Component-Based Systems. Technical Report HPL-2002-25, Imaging Systems Laboratory, HP Laboratories Palo Alto, 2004.

[20] D.B. Petriu and M. Woodside. A metamodel for generating performance models from UML designs. In *Proc. Int. Conf. The Unified Modelling Language: Modelling Languages and Applications*, LCNS 3273, pages 41–53. Springer-Verlag, 2004.

[21] D. Park and S. Kang. Design phase analysis of software performance using aspect-oriented programming. In O. Aldawud, G. Booch, J. Gray, J. Kienzle, D. Stein, M. Kandé, F. Akkawi, and T. Elrad, editors, *The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004*, 2004.

[22] R. Snodgrass. A Relational Approach to Monitoring Complex Systems. *ACM Transactions on Computer Systems*, 6(2):157–196, 1988.

[23] Y. Liao and D. Cohen. A Specificational Approach to High Level Program Monitoring and Measuring. *IEEE Trans. on Software Eng.*, 18(11):969–978, 1992.

[24] J. K. Hollingsworth, O. Niam, B. P. Miller, Z. Xu, M.J.R. Goncalves, and L. Zheng. MDL: A Language And a Compiler For Dynamic Program Instrumentation. In *PACT '97: Proc. 1997 Int. Conf. on Parallel Archit. and Compil. Techniq.*, pages 201–213. IEEE Comp. Society, 1997.

[25] R. Klar, A. Quick, and F. Soetz. Tools for a Model–driven Instrumentation for Monitoring. In *Proc. 5th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 165–180. Elsevier, 1991.

[26] M.B. Blake. A Lightweight Software Design Process for Web Services Workflows. *Proceedings International Conference on Web Services ICWS 2006*, pages 411–418, IEEE Computer Society, 2006.

Claus Pahl
Dublin City University
School of Computing
Dublin 9
Ireland
e-mail: `cpahl@computing.dcu.ie`

Marko Bošković
University of Oldenburg
Software Engineering
D-26111 Oldenburg
Germany
e-mail: `boskovic@informatik.uni-oldenburg.de`

Wilhelm Hasselbring
University of Oldenburg
Software Engineering
D-26111 Oldenburg
Germany
e-mail: `hasselbring@informatik.uni-oldenburg.de`

Author Index