Proceedings of the

International Workshop on Web Services – Modeling and Testing (WS-MaTe 2006)

Palermo, Sicily – June 9, 2006

Preface

Web Services are today the default choice for solving the various issues related to the organization and implementation of software distribution. The attention on this new paradigm is mainly based (a) on the benefits in terms of system interoperability stemming from WSs and (b) on intra- and inter-organisational accounting reasons. Technical solutions are becoming numerous and in the next future we can figure out that the further adoption of common agreed specifications, involving the diverse phases of WS development and description, will raise even more the interest from academia and industries. However, while many current systems are concerned with the migration towards the WS technology, well defined methodologies for modeling service-oriented applications are still lacking, and the impact of this paradigm for what concerns analysis capability remains largely unclear. Aim of the workshop is to put together researchers active in the different aspects of modeling and testing WSs trying to figure out which can be possible reseach solutions.

The idea of organising a Workshop on modeling and testing of Web Services was originated by the research work we carried on within the TELCERT project [1] (FP6 STREP 507128). An ambitious goal of this project is finding new solutions to improve the interoperability of software systems, in particular within the e-Learning domain, in which there is a clear trend towards the adoption of WS technologies. The organisation of the workshop has been mainly possible given the support of the project, whose partners, and especially David Rose and Kevin Riley, we greatly thank.

Much credit for the success of the workshop is due to the program committee members for their thoughtful decisions. A particular thank is due to the staff of the *Istituto di Tecnologie Didattiche* for their support organising the workshop that is hosted in their structure, and to Daniela Mulas and Alberto Ribolini for the technical support.

Finally a special thank to all authors of submitted papers for their interest in WS-MaTe and thanks to the invited speakers and to all workshop participants for making the workshop a lively, friendly (as suggested by the workshop overloaded name) and interesting event.

May 2006

Antonia Bertolino, General Chair
Andrea Polini, Program Chair

[1] http://www.opengroup.org/telcert

Organising Committee

Antonia Bertolino (General Chair)	ISTI/CNR (Italy)
Andrea Polini (PC Chair)	ISTI/CNR (Italy)

Program Committee

Ana Cavalli	INT (France)
Giuseppe Chiazzese	ITD/CNR (Italy)
Giovanni Denaro	University of Milano Bicocca (Italy)
Elisabetta Di Nitto	Politecnico di Milano (Italy)
Wolfgang Emmerich	University College London (United Kingdom)
Istvan Forgacs	4DSoft (Hungary)
Lars Frantzen	Radboud University Nijmegen (Netherlands)
Andreas Holzinger	University of Graz (Austria)
Paola Inverardi	University of L'Aquila (Italy)
Valerie Issarny	INRIA (France)
Mehdi Jazayeri	University of Lugano (Switzerland)
Eda Marchetti	ISTI/CNR (Italy)
Raffaela Mirandola	Politecnico di Milano (Italy)
Manuel Núñez	Universidad Complutense de Madrid (Spain)
Johann Oberleitner	Technical University of Vienna (Austria)
Dumitru Roman	DERI (Austria)
David Rosenblum	University College London (United Kingdom)
Colin Smythe	e-Loki and IMS (United Kingdom)
Jan Tretmans	Radboud University Nijmegen (Netherlands)
Gottfried Vossen	University of Muenster (Germany)
Alex Wolf	University of Lugano (Switzerland)
Fatiha Zaidi	University of Paris Sud (France)
Andrea Zisman	City University (United Kingdom)

Contents

Keynote talk: SOA: Testing and Self-cheking **3**
Gerardo Canfora, Massimiliano Di Penta
(*University of Sannio, Benevento - Italy*)

A Conceptual Model for Adaptable Context-aware Services **15**
Marco Autili, Vittorio Cortellessa, Antinisca Di Marco, Paola Inverardi
(*University of L'Aquila - Italy*)

Modeling of Reliable Messaging in Service Oriented Architectures **35**
László Gönczy, Daniel Varró
(*Budapest University of Technology and Economics - Hungary*)

A Survey on Services Composition Languages and Models **51**
Antonio Bucchiarone, Stefania Gnesi
(*Istituto di Scienza e Tecnologie dell'Informazione/CNR - Italy*)

Towards Model-Based Testing of Web Services **67**
Lars Frantzen, Jan Tretmans, René de Vries
(*Radboud University Nijmegen - The Netherlands*)

Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN **83**
José García-Fanjul, Javier Tuya, Claudio de la Riva
(*University of Oviedo - Spain*)

Initial Investigations into Interoperability Testing of Web Services from their Specification Using the Unified Modelling Language **95**
Colin Smythe
(*eLoki - United Kingdom*)

Detection of Web Service Substitutability and Composability **123**
Michael D. Ernst, Jeff H. Perkins
(*Massachusetts Institute of Technology - USA*)
Raimondas Lencevicius
(*Nokia - USA*)

Towards Efficient Matching of Semantic Web Service Capabilities **137**
Sonia Ben Mokhtar, Anupam Kaul, Nikolaos Georgantas, Valerie Issarny
(*INRIA Rocquencourt - France*)

Keynote Talk

SOA: Testing and Self-cheking

Gerardo Canfora and Massimiliano Di Penta
University of Sannio - Benevento (ITALY)

SOA: Testing and Self-Checking

Gerardo Canfora and Massimiliano Di Penta [1]

RCOST - Research Centre on Software Technology - University of Sannio
Palazzo ex Poste, Via Traiano - 82100 Benevento, Italy

Abstract

The dynamic nature of service-oriented architectures poses new challenges to system validation. Traditional testing is unable to cope with certain aspects of a service–oriented system validation, essentially because of the impossibility to test all (often unforeseen) system's configurations. On the other hand, run–time monitoring, while able to deal with the intrinsic dynamism and adaptiveness of a service–oriented system, are unable to provide confidence that a system will behave correctly before it is actually deployed.

In this paper we discuss the role of testing and monitoring to validate a service–oriented system and how they can be combined to increase the confidence and reduce the cost of validation.

Key words: Web Services, Service–Oriented Architecture, Service Testing, Service Monitoring

[1] Emails: {canfora, dipenta}@unisannio.it

1 Introduction

Testing software has long been recognized as a key and challenging activity of system development. Service–Oriented Architectures (SOA), for example as implemented by web services, present unique features, including dynamic and ultra–late binding, that add much complexity to the testing burden. A few notable examples are such features are:

- systems based on web services are intrinsically distributed, and this requires that Quality of Service (QoS) be ensured for different deployment configurations;
- web services in a system change independently from each other;
- systems implement adaptive behaviors, either by replacing individual services or adding new ones;
- ownership over the system parts is shared among different stakeholders.

Many consolidated testing approaches, applied for years over traditional systems, apply over service–oriented systems as well. Primarily, the idea that a combination of unit, integration, system, and regression testing is needed to gain confidence that a system will deliver the expected functionality. Nevertheless, the dynamic and adaptive nature of SOA makes most testing techniques not directly applicable to test services and service–oriented systems. As an example, most traditional testing approaches assume that one is always able to precisely identify the actual piece of code that is invoked at a given call-site. Or, as in the case of object-oriented programming languages, that all the possible (finite) bindings of a polymorphic component be known. These assumptions my not be true anymore for SOA, which exhibit run-time discovery in an open marketplace of services and ultra–late binding.

The adoption of SOA, in addition to changing the architecture of a system, brings changes in the process of building the system and using it, and this has effects on testing too. Services are used, not owned: they are not physically integrated into the systems that use them and run on a provider's infrastructure. This has several implications for testing: code is not available to system integrators; the evolution strategy of a service (that is, of the software that sits behind the service) is not under the control of the system owner; and, system managers cannot use capacity planning to prevent QoS failures.

Key issues that limit the testability of service–oriented systems include [6]:

(i) *lack of observability of the service code and structure:* for users and system integrators services are just interfaces, and this prevents white–box testing approaches that require knowledge of the structure of code and flow of data.

(ii) *dynamicity and adaptiveness:* for traditional systems one is always able to determine the component invoked in a given call–site, or, at least, the set of possible targets [9]. This is not true anymore for SOA, where a system can be described by means of a workflow of abstract services that are automatically bound to concrete services retrieved by one or more registries during the

execution of a workflow instance.

(iii) *lack of control:* while components/libraries are physically integrated in a software system, this is not the case for services, which run on an independent infrastructure and evolve under the sole control of the provider. The combination of these two characteristics implies that system integrators cannot decide the strategy to migrate to a new version of a service and, consequently, to regression testing the system [5].

(iv) *cost of testing:* invoking actual services on the provider's machine has effects on the cost of testing, too, if services are charged on a *per–use* basis. Also, service providers could experience denial–of–service phenomena in case of massive testing, and repeated invocation of a service for testing may not be applicable whenever the service produces side effects other than a response, as in the case of a hotel booking service [6].

An alternative to testing is continuous self-checking of a service–oriented system by monitoring it during execution [2]. Runtime monitors can check both the functional correctness and the satisfaction of QoS expectations, thus realizing the idea of continuous testing [11]. Being performed at run-time, that is, on the actual running configuration of the system, monitoring can naturally accommodate the intrinsic dynamicity and adaptability of SOA. It does not have cost problems, or undesired side effects on real world, as the service invocation issued are those actually needed to run the system. Of course, there is some run–time overhead, but this is acceptable in most cases and, in addition, modern monitoring infrastructures allow for setting the amount of monitoring at run–time [2].

However, self–checking has limitations, too. The system is checked during actual execution, and this may entail that exceptional conditions remain unchecked, including peak usage and scarcely recurrent function or combinations of input data. More importantly, problems are discovered too late, when the system is executing. Whilst several recovery actions are possible [2, 7, 8], it is not always possible to rely solely on these actions, as in the case of dependable applications. There are many cases when a system needs to be validated before deployment, and in these cases monitoring does not apply.

Thus, on the one side traditional testing has limitation to deal with the dynamic nature of SOA, primarily because of its inability to foresee changes, and on the other side self–checking does not help to assess the quality of a system prior of its use. We believe that testing and monitoring are two complementary facets for validating a service–oriented system and, in this paper, we discuss how they complement each other in the process of gaining confidence of the correctness of a system over time.

The rest of this paper is organized as follows: Section 2 discusses the different roles played by testing and monitoring in SOA. Section 3 highlights the need for having both testing and monitoring and discusses how each one can benefit from the other. Finally, Section 4 summarizes the paper and outline directions for future research.

2 The Role of Testing and of Monitoring

As mentioned in the introduction, service functional and non functional characteristics can be either tested before making the service operational, or monitored at run–time. As it will be described below, the roles of testing and monitoring are different.

To better understand how things work in the practice, let us consider a service–oriented system – realized for example as a BPEL process – that performs image processing. The process (see Figure 1) comprises several services realizing different tasks, i.e. image scaling, posterizing, sharpening, or reduction of colors to a gray scale.

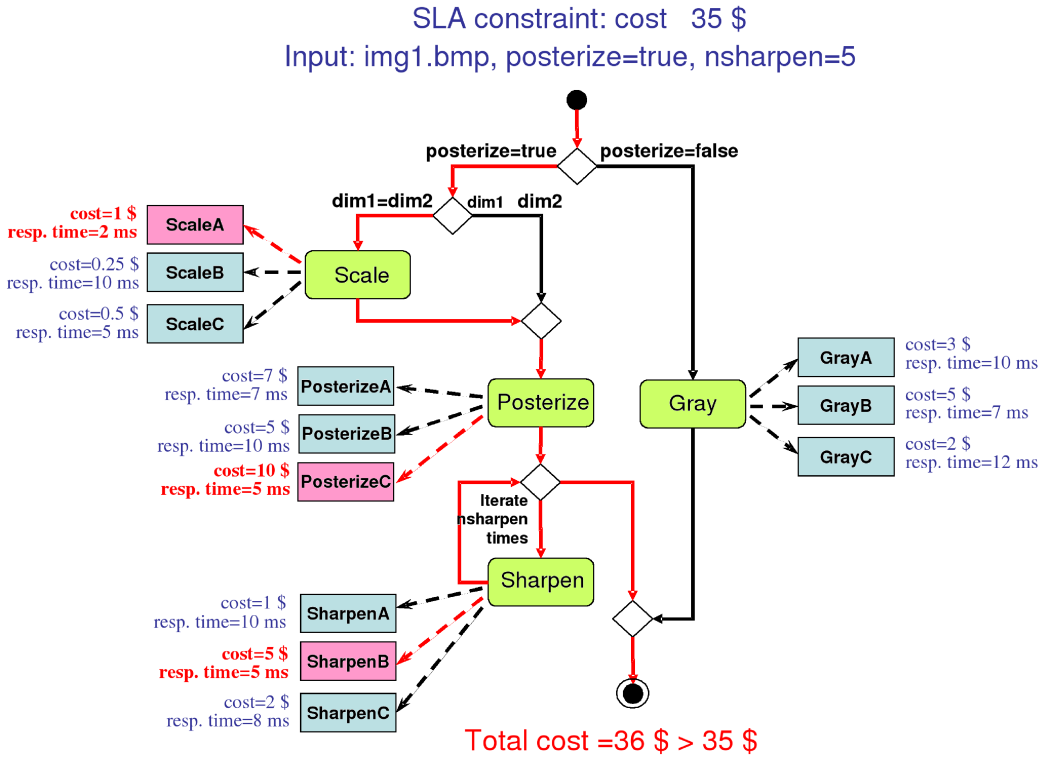


Fig. 1. Running example: image transformation process

The remainder of this section describes how different service characteristics – namely functional aspects, dynamic aspects, and, finally, the way a service evolves – can be tested or monitored, highlighting the role of testing and monitoring.

2.1 Checking the service functional properties

2.1.1 The role of testing

Except when a service is tested by its developer, that has the source code available (whilst the configuration could be far from those where the service will be actually deployed) black box testing is the only viable solution. A possibility is to perturbate SOAP messages to check whether the service is robust to these perturbations, or if

the effect of the perturbation is observable from the service response [10]. For the image processing service, perturbation of parameters – e.g., different *nsharpen* or *posterize* values – should be observable from the different image produced as output or from the error message possibly generated.

Another possibility is to rely on types defined as XML schema within the WSDL to generate test cases [1, 12]. For example, inputs for our example would lead to equivalence classes proper of image processing option parameters (e.g., `true` or `false` for *posterize*, and six classes for *nsharpen*, i.e. $nsharpen = 0$, $nsharpen = 1, 1 < nsharpen < max\_nsharpen-1, nsharpen = max\_nsharpen-1$, $nsharpen = max\_nsharpen$ and $nsharpen > max\_nsharpen$).

2.1.2 The role of monitoring

Instead of testing the services, some monitoring approaches aims at checking whether some post-conditions are met after the service execution [2, 8]. If this does not happen, proper recovery actions are taken. Monitoring rules are often weaved as crosscutting concerns across the system source code (or across its BPEL process). For example, if a image scaling service is not able to handle an image over a given size, either the process execution must be aborted or a different binding has to be chosen.

2.2 Checking the service QoS

2.2.1 The role of testing

When a service is acquired by a consumer s/he stipulates a Service Level Agreement (SLA) with the provider, which comprises the specification for the QoS level that the provider will ensure to the consumer. At run time, QoS constraint violation can be due to the environment (high network traffic, high number of requests), but also to unexpected inputs. For example, let us suppose that the provider of the *Posterize* service states that images below 2 MB can be processed within 10s. It might happen that, for some posterizing preferences, such a constraint cannot be met. Things get worse when the service under test is composite, and different bindings can lead to different QoS.

The role of testing is therefore to generate combinations of inputs and bindings that cause a violation of QoS constraints. The example in Figure 1 shows how for the image `img1.bmp`, $posterize = true$, $nsharpen = 5$ and the abstract services *Scale*, *Posterize* and *Sharpen* bound to *ScaleA*, *PosterizeC* and *SharpenB* the constraint $cost < 35\$$ is violated.

2.2.2 The role of monitoring

QoS violations can also be handled at run–time using a monitoring mechanism that triggers service re–binding actions. After the (near) optimal set of binding has been determined, the service oriented system starts its execution. New QoS estimates made at run–time, or the lack of availability of a service, can trigger the re–planning of services still to be executed [7]. This will allow to meet QoS

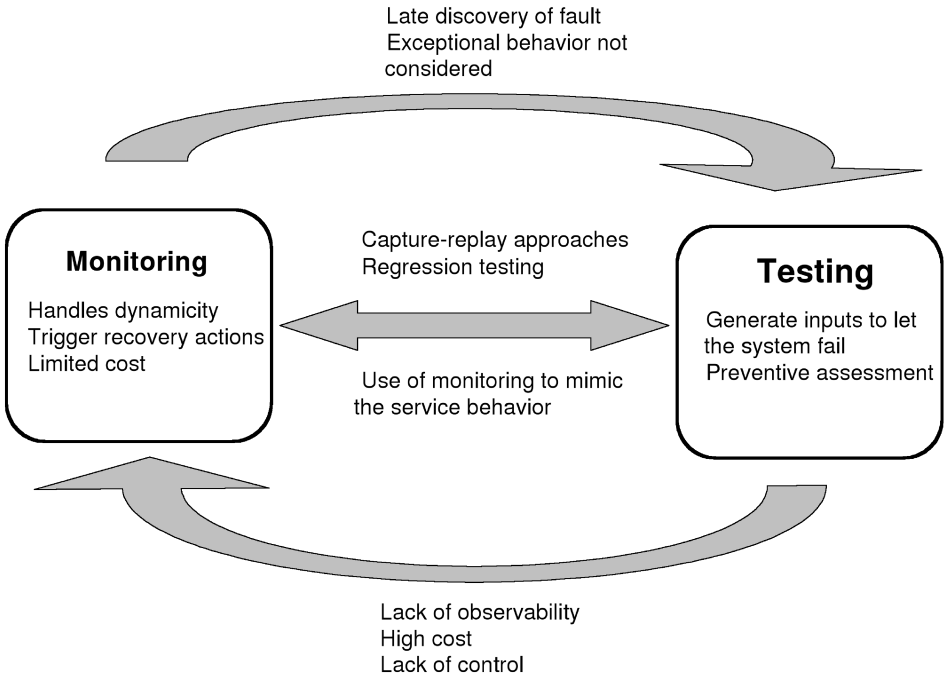


Fig. 2. The role of testing, of monitoring and their interaction

constraints that would have been otherwise violated, and to improve the overall QoS objectives as well (e.g., minimizing the cost or the response time).

Let us assume that, during the execution of the image processing service, the actual, monitored response time for the scaling service is higher than what previously estimated. This causes an increase of the overall service response time, leading to a potential constraint violation. To avoid this, the services still to be executed (i.e., *posterize* and *sharpening*) will be bound to faster (even if more expensive) concretizations that allow response time constraints to be satisfied.

2.3 Checking service interoperability

2.3.1 The role of testing

Interoperability check can vary from the simple check of the compliance to WS-I [2] to the integration testing of services composed in a process or service–oriented system. To test service interoperability, the UDDI registries can change their role from a passive role of service directory to an active role of accredited testing organism [4].

As described in reference [6], dynamic binding makes interoperability issues more and more complex. In such a context, integration testing becomes very expensive, since any service invocation within a process or system should be tested against any possible binding. Problems due to polymorphism in object–oriented systems [9] tend to explode here, even because, very often, bindings are not known

[2] *http://www.ws-i.org*

a priori.

Ideally, the process in our example should be tested for $3 \cdot 3 \cdot 3 \cdot 3 = 81$ possible combinations of concretizations. In the practice, it would suffice to only test those combinations that are compatible with our global QoS constraints.

2.3.2 The role of monitoring

Integration problems can be checked at run–time using monitoring mechanisms. When a binding changes, the new end–point should preserve the post–condition held for the old end–point. If this does not happen, an alternative service should be chosen. For example, if re–binding chooses a new *Posterizing* service that produces an image violating some post–conditions met before – e.g., color depth > 24 bits – then this service should be discarded in favor of an another.

2.4 Checking the service evolution

2.4.1 The role of testing

Regression testing is essential for service–oriented systems, since integrators are out of control of the service being used. When a service evolves, its functionality or QoS can vary, impacting over the systems using it. This raises the need for service regression testing: as proposed by Bruno *et al.* [5], services needs to be accompanied with a facet, containing test suites that the integrator can use for the regression testing of the service functional and non functional properties, either periodically or when a new release of the service has been issued.

For example, the *Sharpen* service implementation can change: either the resulting image can be different, or the response time can be larger than the one experienced when the service was acquired and the SLA negotiated.

2.4.2 The role of monitoring

Monitoring assumes an important role for checking the evolution of a service–oriented system. Once again, a post–condition checking mechanism can be used to check whether the service, while evolving, continues to preserve the functional and non functional properties the integrator is expecting.

3 Combining Testing and Monitoring

The previous section described benefits of different testing and monitoring approaches. One can argue whether it could be possible to avoid testing service–oriented systems and just perform run–time monitoring, followed by proper recovery actions. On the other hand, it can be decided that if a system has been properly tested, monitoring is not needed.

Nevertheless, due to the different roles assumed by testing and monitoring (see Figure 2), we often need both:

(i) Testing is a preventive activity, to be performed before delivering (or before

using) the service. Also, testing exercises the service with the objective of discovering faults. This goes beyond from checking the correctness of the regular service usage, addressed by monitoring.

(ii) Monitoring is performed at a different time, i.e., after the service has been executed. Whilst monitoring can trigger recovery actions, in some cases it may be too late to do anything useful. If the image processing service has just violated its response time constraints, nothing can be done to recover such a violation; the service should have been tested before to check its ability to guarantee a given response time for any allowed input configuration.

As the figure shows, there are many weaknesses of monitoring that suggest to perform testing, and *vice versa* (see the arrows between the monitoring and testing boxes). Also, there are many cases where monitoring strategies can be combined with testing (double arrow). For example, other than using test cases made available with the service, regression testing can be performed using capture–replay strategies. The service inputs and outputs can be monitored and, after the service has evolved, inputs are replayed and outputs observed.

The cost of regression testing can be reduced by using monitoring data for building service stubs that simulate the response to requests equal or similar to those recently made by other users [6].

Service QoS testing strongly relies on monitoring mechanisms that, at minimum, are used to measure the QoS related to a service execution during testing activities. Once again, stubs built upon monitoring data can be used to limit the number of service invocations required during the testing phase.

When replacing the end–point of an abstract service, data monitored from the previous end–point can be used to test the new one, to ensure that it properly interoperates with our system. For example, if the *PosterizeA* end point is replaced by *PosterizeB*, it is worth using monitoring data from *PosterizeA* to test *PosterizeB* and check whether the new end point preserves the current behavior.

Finally, service functional testing can benefit of monitoring mechanisms. Monitors can be used to implement oracles (i.e., by checking post–conditions).

4 Summary

The paper has discussed the complementary roles of testing and run–time monitoring in the process of validating service–oriented systems. On the one hand, run–time monitoring is needed to deals with systems configurations that change in an unforeseen way; on the other hand, testing is still indispensable to gain confidence on the correctness of service–oriented systems before they are actually deployed for use.

Of course, both testing and monitoring of service–oriented systems present open problems that need further research. A key issue of monitoring is to balance the degree of run-time checking and the impact on the performances of the system. Ideally, monitors should allow for setting the amount of monitoring at run–time

based on the needs of single users [2]. Also, it is needed that monitors be able to work with existing standard technologies, such us standard BPEL engines.

The idea itself of integration testing is challenged by SOA unique features, primarily automated search on services in an open space and ultra–late binding. These features implies that the actual configuration of services involved in a system's run for a given user be known only at execution time. In many cases, however, however, searching is limited to a bounded space, as for example when it is required that a contract be signed before a service can be used. In these cases, conservative approaches that test possible system configurations while minimising the number of test runs are needed.

QoS testing poses new challenge, too. Services run on the (foreign) infrastructures of providers and payment may be on a per-use basis, which makes stress testing prohibitively costly. Monitoring data from past execution could help reducing the cost of testing by minimising the needs for actual calls to services.

5 Acknowledgments

This work is framed within the European Commission VI Framework IP Project SeCSE (Service Centric System Engineering) (*http://secse.eng.it*), Contract No. 511680.

References

[1] X. Bai, W. Dong, W.-T. Tsai, and Y. Chen. In *WSDL-Based Automatic Test Case Generation for Web Services Testing*, pages 215–220, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[2] L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. In Benatallah et al. [3], pages 269–282.

[3] B. Benatallah, F. Casati, and P. Traverso, editors. *Service-Oriented Computing - ICSOC 2005, Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005, Proceedings*, volume 3826 of *Lecture Notes in Computer Science*. Springer, 2005.

[4] A. Bertolino and A. Polini. The audition framework for testing Web services interoperability. In *EUROMICRO-SEAA*, pages 134–142. IEEE Computer Society, 2005.

[5] M. Bruno, G. Canfora, M. Di Penta, G. Esposito, and V. Mazza. Using test cases as contract to ensure service compliance across releases. In Benatallah et al. [3], pages 87–100.

[6] G. Canfora and M. Di Penta. Testing services and service-centric systems: Challenges and opportunities. *IT Professional*, 8(2):10–17, 2006.

[7] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. QoS-Aware Replanning of Composite Web Services. In *in Proc. of the International Conference on Web Services (ICWS 2005)*, Orlando, FL, USA, July 2005.

[8] K. Mahbub and G. Spanoudakis. A framework for requirents monitoring of service based systems. In M. Aiello, M. Aoyama, F. Curbera, and M. P. Papazoglou, editors, *ICSOC*, pages 84–93. ACM, 2004.

[9] A. Milanova, A. Rountev, and B. G. Ryder. Parameterized object sensitivity for points-to analysis for java. *ACM Trans. Softw. Eng. Methodol.*, 14(1):1–41, 2005.

[10] J. Offutt and W. Xu. Generating test cases for web services using data perturbation. *SIGSOFT Softw. Eng. Notes - SECTION: Workshop on testing, analysis and verification of web services (TAV-WEB)*, 29(5):1–10, 2004.

[11] D. Saff, S. Artzi, J. H. Perkins, and M. D. Ernst. Automatic test factoring for java. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 114–123, New York, NY, USA, 2005. ACM Press.

[12] W. T. Tsai, X. Wei, Y. Chen, and R. Paul. A robust testing framework for verifying web services by completeness and consistency analysis. In *IEEE International Workshop on Service-Oriented System Engineering (SOSE)*, pages 159–166, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

Session 1: Modeling

Paper 1
A Conceptual Model for Adaptable Context-aware Services
Marco Autili, Vittorio Cortellessa,
Antinisca Di Marco, Paola Inverardi
(*University of L'Aquila - Italy*)

Paper 2
Modeling of Reliable Messaging in
Service Oriented Architectures
László Gönczy, Daniel Varró
(*Budapest University of Technology and Economics - Hungary*)

Paper 3
A Survey on Services Composition Languages and Models
Antonio Bucchiarone, Stefania Gnesi
(*Istituto di Scienza e Tecnologie dell'Informazione/CNR - Italy*)

A Conceptual Model for Adaptable Context-aware Services

Marco Autili, Vittorio Cortellessa,
Antinisca Di Marco, Paola Inverardi [1,2]

Dipartimento di Informatica, Università dell'Aquila
I-67010 L'Aquila, Italy

Abstract

The rapid evolution of (Web) service technologies is leading services to play a central role in the software development process. The number of service-based applications is quickly increasing, and they are subject to continuous evolution to meet ever more demanding requirements, such as: adaptability to different types of devices, optimal exploitation of available resources to achieve a certain level of Quality of Service. In addition, Service Oriented Architectures are replacing Software Architectures as a main software model that allows to represent and validate functional and non-functional properties of the system. This paper content originates from the basic goal of the PLASTIC project, where we work to provide tools and methodologies to develop service-based applications that are adaptable to the context and able to offer the best tradeoff between offered QoS (from the platform) and required QoS (from users). The first step towards this goal consists in building a common dictionary. Here we propose a first version of the PLASTIC dictionary, that is a conceptual model that embeds three original aspects with respect to other existing service models: (i) relationships between components and services are well defined while keeping feasible the composability in both domains, (ii) context is defined and related to the other concepts of the model, (iii) QoS characteristics do not explicitly appear, but are scattered among several concepts. The latter two aspects induce adaptability to the context in the services built within our conceptual model.

Key words: Service, Software Service, Component Orientation, Service Orientation, Service Conceptual Model.

[1] This work is supported by the PLASTIC project: Providing Lightweight and Adaptable Service Technology for pervasive Information and Communication. Sixth Framework Programme. http://www.ist-plastic.org

[2] Emails: {marco.autili, cortelle, adimarco, inverard}@di.univaq.it

1 Introduction

Software pervades our life, at work and at home, spanning from business to entertainment. We increasingly expect it to be dependable and usable, despite of our own mobility and changing needs.

Ubiquitous networking makes possible for mobile users to access networked software services across heterogeneous infrastructures. However even if connectivity is provided, heterogeneity poses numerous challenges to software developers, among which we mention: developing services that can be easily deployed on a wide range of evolving infrastructures, from networks of devices to stand-alone wireless resource-constrained hand-held devices; making services resource-aware so that they can benefit from networked resources and related services; and ensuring that users meet their non-functional requirements by experiencing the best Quality of Service (QoS) possible according to their needs and specific contexts of use.

As pointed out in [2], while delivering services, applications need to be adaptive to a context that is the combination of user-centric data (e.g., information of interest for the user according to his/her current circumstance) and resource/computer-centric data (e.g., resource constraints and conditions of the user device). Additionally, services must be provisioned in a way that guarantees their dependability.

Adaptable applications are software applications that can be adapted with respect to the environment that will host their execution. In order to perform an adaptation it is essential to provide an actual way to model the characteristics of the application itself, of the heterogeneous infrastructures and of the execution environment including the end-user degree of satisfaction. In such a domain, it is useful to be able to reason about the resources required by an application (and its possible adaptations) and the ones supplied by the hosting environment.

In this paper we describe an initial service conceptual model that is the basis of a framework for the development, the deployment and validation of adaptable software services targeted to mobile and possibly limited devices (e.g., smart phones, PDAs, etc.) and running on heterogeneous network infrastructures.

Targeting adaptive services, this conceptual model focusses on the concept of context of use and related Service Level Agreement (SLA). We introduce the notion of *Service Request Requirements* to address the preferences of the user that will be used to establish the SLA between the customer and the provider of the service. This work is carried on in the context of the PLASTIC project [16].

Recently, several approaches to conceptualize the world of services have been proposed. Our model is inspired by the SeCSE conceptual model [12], developed within the context of the SeCSE project [18]. It is very general since it does not focus on specific service domains. Moreover it is extensible, and this feature allowed us to customize it to the needs of the PLASTIC project.

In the following, before introducing the description of the conceptual model, we discuss the meaning of service in real-world and in real-life contexts and how this is reflected in the software domain. Then we discuss differences and similarities of

component orientation and service orientation. This issue is particularly relevant in our context since PLASTIC services rely on component-based applications. As already proposed in literature, these two approaches can be combined in order to produce service oriented component models.

To this purpose, it is useful to point out the set of principles that a Service Oriented Architecture (SOA) should have. Moreover, even though PLASTIC concentrates on service development (i.e. it mostly works on the service developers and service providers side), we will also discuss the roles played by the service registry, service provider and the service consumer entities within the well known *Service Oriented Interaction Pattern* for service publication, discovery and binding.

We remark that this work reports our initial attempt to conceptualize the world of services within the PLASTIC project. However, it is worth to note that the resulting conceptual model is suitable for any domain where adaptation with respect to QoS and context characteristics is central.

The paper is structured as follows. Section 2 provides a first flavor of PLASTIC project vision; in section 3 we introduce basic concepts of Service Oriented Architectures, while in section 4 we give our two-layers vision of service orientation vs component orientation; section 5 introduces the original aspects that characterize our conceptual model; finally, in section 7 we discuss the related work, and in section 8 the concluding remarks are given.

2 PLASTIC project: the vision

In this section we briefly describe the PLASTIC vision to provide a possible context where the conceptual model could be used.

The goal of the PLASTIC project is the construction of a platform for delivering software services deployed over B3G [1] networks. Service development will have its foundations on the service-oriented and component-based paradigms.

In particular PLASTIC will take advantage from Web Services (WS) and more standard component-based technologies. From the PLASTIC point of view an application is seen as the composition of autonomous networked services, and an individual service is developed by assembling software components.

Networked software services are deployed over the heterogeneous open wireless environments empowered by B3G networks. Services will be adaptive to the environment with respect to resource availability and delivered Quality of Service (QoS), via a development paradigm based on Service Level Agreements (SLAs) and resource-aware programming. The context-of-use adaptation that will be addressed in PLASTIC is two-fold: from the service provider side and from the user side.

The most interesting aspect is that, across the different layers, the relevant characteristics of the various heterogeneous infrastructures will be abstracted in models and made observable at the application layer. In this way applications can manage service adaptation to a greater degree of flexibility.

3 Services and Service Oriented Architecture

Bringing together the vision of the academic world and the vision of SUN [19], IBM [11] and Microsoft [6] about Service Oriented Architecture (SOA), in this section we discuss the basic question arising in the world of services and describe the SOA vision by highlighting principles and primary involved entities.

In the survey [5] the authors collect and classify a set of terms related to the concept of service. They distinguish between terms - such as IT-services, information services, public services, governmental services, etc - that consider domain-related *contents* of the service and terms - such as service, e-service, Web services, Real-World services, commercial services, etc - that are called "*service terms*" and that strictly relate to the *definition* and *essence* of a service itself.

Focussing on service terms, the authors point out how these terms have multiple interpretations for the different communities of *business* science, *information* science and *computer* science, resulting in confusion. That is, starting from the awareness of the existence of differing terminologies among different research communities, they explain how these different interpretations are related and how can be combined in order to achieve a common understanding and a shared terminology for a profitable collaboration.

Concentrating on the terms *services*, *e-service* and *Web-service* (i.e., most commonly used service terms) and surveying visions of the above mentioned communities, the authors collect definitions for these terms.

The business science community make use of the term services (with no prefix) to refer "*business activities that often result in intangible outcomes or benefits; they are offered by a service provider to its environment*". The same community gives three different - yet similar - interpretations for the term e-service. According two of them, an e-service is "*an Internet-based version of a traditional service*": one interpretation is more extensive since it mentions customer relationships or business processes; the other one does not. The third interpretation includes the first one but it is much extensive since it is not limited to the Web but it allows for the "*provision of service over generic electronic networks*".

Web service is not considered a business term, and when it is used one of the definitions from the computer science community is adopted. Many definitions exist within the computer science community for the term Web Service. Bringing together elements such as software/applications, software functionalities and Internet, in [5] the authors realize how all of the definitions agree on the fact that Web services are software/applications to be used on the Internet. Most of them explicitly recognize "*the existence of functionalities behind the software but not the existence of business processes or business functionalities*".

There is no general agreement for the term e-service. Sometimes it refers to "*the realization of federated and dynamic e-business components in the Internet environment*" but often it is considered a Web service synonym. Moreover, the term service is also used as a synonym for both Web service and e-service. Some, on the other hand, give to the term service a business-based interpretation defining

it as an intangible product, benefit, good.

Finally, in [5] the authors conclude the comparison by stating that, for the information science community, the term Web service is used like in computer science and the term service is mostly used like in business science. In the same manner, e-services are interpreted either as an Internet-based version of traditional services (similar to many researchers from business science), or as Web services (similar to many computer scientists). Other terms such as commercial service and real-world service are sometimes used to refer to services in their business meaning.

Taking the W3C Web Services Architecture (WSA) [20] as important input, in [17] the author presents an abstract conceptual architecture for semantic web services and, by analyzing a set of case studies, defines requirements on the architecture. Even though the author concentrates on semantic web-service, it is interesting the discussion made on how the word service can be used in several different ways in different domains.

Recognizing the essentiality of three main aspects such as (i) "value" in some domain, (ii) implementation of entities able to provide what is needed, and (iii) interactions involved, three usages of the word service are given.

(i) Service considered as provision of value in some domain: distinguishing between the particular provision of value itself from the general capability to provide, the author refers to the former as a concrete service and to the latter as an abstract service. According to this distinction, "*a concrete service is the provision of something of value, in the context of some application domain, by one party to another*"; "*an abstract service is defined as the capacity to perform something of value, in the context of some application domain*".

(ii) Service considered as a software entity able to provide something of value: commonly used in the computer science and IT community, in this domain the author prefers to define a service as "*a software entity being (part of) a service provider agent*". Rather than to speak about sending messages to services, receiving results from services, executing services etc, the author prefers to maintain generality in order to avoid confusion when mixing the usage of this definition with the previous definition.

(iii) Service as a means for interacting online with a service provider: this usage is referred as the "*provision of a negotiation protocol or choreography*". Since the negotiation is not itself a value, on one hand, this usage is different from the usage (i) but, since the outcome of the negotiation may be a value, on the other hand, this usage may be a service in the sense (i).

According to the definition given from the European Commission [3], one of the more abstract and context independent definition of services might be: "*Services are labors that, if accomplished, produce a non tangible useful benefit*". Still according to the definition given from the European Commission [3], *Software Services are functionalities provided by software applications that supply computational resources or informational ones on request performed by an hypothetical service consumer*. The interpretation that we give to this definition is that an hypothetical service consumer can use the provided functionalities for exploiting com-

putational resources and for obtaining informational data, both of them abstracted in software services. In other words functionalities are abstracted by services.

The potential of service oriented applications has brought to introduce in the software development domain (beside the Software Architecture view) the definition of SOA. However, in order to be an architecture model, the *service domain* needs more than just service descriptions: the overall application dynamics has to be described by a workflow between services.

Generally speaking, Software Architectures (SA) are meant to abstractly model the overall structure of a system in terms of communicating *subsystem*. In a component based setting, this general definition of SA can be rephrased by substituting the word *subsystem* with *set of software components* interacting through communication channels. In a service oriented vision, SOA are meant to abstractly model the overall structure of service-centric software systems in terms of communicating *services*. They are a special kind of software architectures that have several unique characteristics according to a set of principles that we later discuss.

Alternatively to the traditional tightly-coupled object-oriented models that have emerged in the past decades, SOA is an alternative loosely-coupled model. Moreover, the most significant aspect of service-oriented architecture is that it separates the service implementation and the service provider from its contractual description. In this way, it separates the "WHAT" from the "HOW". The service description is given in neutral manner, independently from any platform and implementation detail. Service consumers view a service simply as an endpoint that supports a particular request format or contract. Service consumers do not need to know how the service will act for satisfying their requests; they expect only that it will.

Due to the rapid evolution of software service technologies, the development of service-based software applications can lay nowadays on a solid support and is adopted in ever wider application domains. Applications are built as assemblies of existing services, and ever more mature tools for discovering (over the web) and adapting services are being developed.

According to the technical protocols and business model used, the European Commission [3] classifies service saying that Web Services, Grid Services, P2P Services and SOA are classes at the same level. In our vision, Service Oriented Architectures are the super class of all the other classes since it is the most general, business model independent protocol.

SOA is not new. A classic example of a proto-SOA system that has been around for a while is CORBA, which defines similar concepts to SOA. SOA is different from it in the roles played by "interfaces" (description). In fact, SOA trusts in more flexible and neutral language for describing interface. If we consider Web Services (WS) as a particular implementation based on SOA, then the description of interfaces in an XML-based language (called WSDL [7]) has moved WS to a more dynamic and flexible interface system than the older IDL of CORBA.

Web services aren't the only way to implement SOA. *Message-Oriented Middleware* systems, such as the IBM MQ Series [10], represent an alternative and, as just mentioned earlier, CORBA was a first attempt. The difference is on "how

well" they meet the "directives" of the SOA philosophy.

4 Component Orientation VS Service Orientation

In this section we firstly describe the main peculiarities of component and service oriented approaches starting from the work about service oriented component model in [9]. Then, we describe how these approaches can be combined in order to obtain a two-layer framework that provides infrastructures for building service-centric and component-based applications while keeping composability in both domains. We also discuss how such an approach is suitable for the PLASTIC development platform.

4.1 A classical view

Starting from an assembly-oriented view of software engineering, CBSE [4] promotes the construction of a software application as composition of reusable building blocks called components. Assemblers build applications by wiring ports of connectors and "ready to use" components. Hence, integration is carried out at assembling time. As a direct consequence, composition is structural in nature and the component should provide the external structure of its instances (such as provided and required functional interfaces). Preconditions, post-conditions, and a behavioral specification of the interaction with its environment may be also required to allow assemblers to connect and eventually adapt the various component instances. Hierarchical structural composition is achieved when the external view of a component is itself implemented by a composition.

Unfortunately, software components are often hardly reused and Component Adaptation (CA) is emerging as a well-differentiated discipline. CA focuses on the problem of reusing existing software components when constructing a new application, and mostly promotes the use of adaptors (specific computational entities) for solving this problem. Software adaptors should guarantee that components will interact in the right way not only at the signature level, but also at the interaction protocol and semantic levels.

Our experience [13, 15] on component adaptation is based on the automatic synthesis of local adaptors (one for each component) that constitute a decentralized and possibly distributed adaptor for the component based system. The approach acts at the interaction level and the intent of the distributed adaptor is to moderate the communication among the components in a way that the system complies only to a specific behavior.

On the other end, resource awareness identifies the capability of being aware of the resources offered by an execution environment, in order to decide whether that environment is suited to receive and execute the application. In this concern, adaptation identifies the capability of changing the application in order to comply with the current resource conditions of the execution environment. In [8] we propose a framework for the development and deployment of adaptable component-based

software systems.

Sharing the same idea of component orientation, in service orientation software applications are assembled from reusable building blocks, but in this case the blocks are represented from services. Each service has a description of provided functionalities that can be contractually defined as a mix of syntax, semantics and behavior specifications. The basis for service assembling are only service descriptions. In this way, differently from the component orientation, the service integration may be carried out prior or at run time. In such a context, it is clear that service orientation concentrates on how the service are described in order to support dynamic discovery and possibly run-time integration.

In order to understand how dynamic discovery may be supported, we describe the *Service Oriented Interaction Pattern* (see figure 1) and the roles played by the service registry, service provider and the service consumer entities.

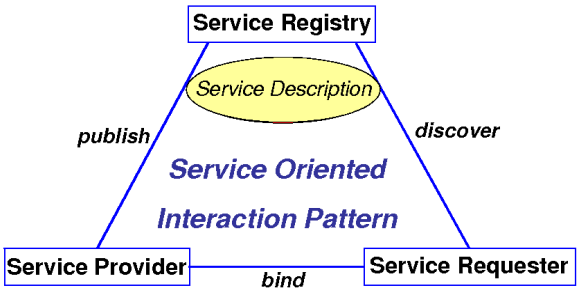


Fig. 1. Service Oriented Interaction Pattern

Service providers publish their service descriptions into a service registry. According to the service request format, service requesters query the service registry to discover service providers for a particular service description. If one or more providers are present in the service registry at the moment of the request, the service requester can select and bind to any of them. When a service requester binds to the service provider, the latter returns a reference to a service "object" that implements the service functionality.

Service orientation promotes the idea that a service may be supplied by different service providers. In fact whenever other providers comply with the contract imposed by the service description they can be interchanged. In this way a requester is not coupled with a particular provider.

Another fundamental characteristic of service orientation is that services exhibit dynamic availability, since they can be added or removed from the service registry at any time. Consequently, running applications must be capable of releasing departing services or of incorporating newly arriving services.

Since services composition is based only on service description, it can be seen as an abstract composition that become concrete only at run-time.

Table 1 summarizes in a point-to-point view the main aspects of service orientation and component orientation.

The growing request of dynamic and mobile software systems able to exploit

Service Orientation	**Component Orientation**
Emphasis on software service descriptions.	Emphasis on component external-view.
Natural separation from description and implementation.	No evident separation between component external-view and implementation
Only service description is available during assembly.	Component physically available during assembly.
Integration is made prior or during execution.	Integration at assembly-time.
Better support for run-time discovery and substitution.	No native support for run-time discovery and difficult substitution.
Native dynamic availability.	No native dynamic availability.
Abstract composition.	Structural composition.

Table 1
Main aspects of Service Orientation and Component Orientation

available resources and to adapt to them in order to achieve a certain level of QoS has pointed out several limits of the component orientation for this type of software applications. Component orientation does not provide mechanisms crucial for mobile and adaptable applications and needed to evolve the system at run-time. As reported in the table, component orientation (i) has no native support for dynamic discovering and availability and (ii) it allows easy integration only at the assembly-time that makes dynamic component substitution very difficult to implement. These limitations are strictly related to the fact that there is no evident separation between components description and its implementations.

Service orientation instead overcomes these limits by providing useful mechanisms for dynamic evolution. However, the service orientation, differently from the component orientation, is more abstract. In fact, it does not describe or give indications on how services should be implemented, leaving the development freedom to chose about the way he may implement the software services. On the other hand, the component orientation allows excellent structuring of the implementation code easing the integration and the maintenance of available components.

We believe that the component and the service orientation can be combined to overcome the limits and the lacks of both development approaches and we discuss in the following section how such combination can be realized.

4.2 Our view: a two-layers approach combining services and components

In this section we present the basis of the service-oriented component model that we are building for the PLASTIC platform. Modularization and separation of concerns (i.e., service description and implementation) are the driving principles of this model.

Figure 2 shows our two-layers vision of a software application. The layers are: *service layer* and *component layer*.

The component layer represents the computing environment for networked services that is able to manage the life cycle of the software components while delivering functionalities for a device from anywhere in the network. Software components can be deployed, updated, or removed on the fly without interrupting the

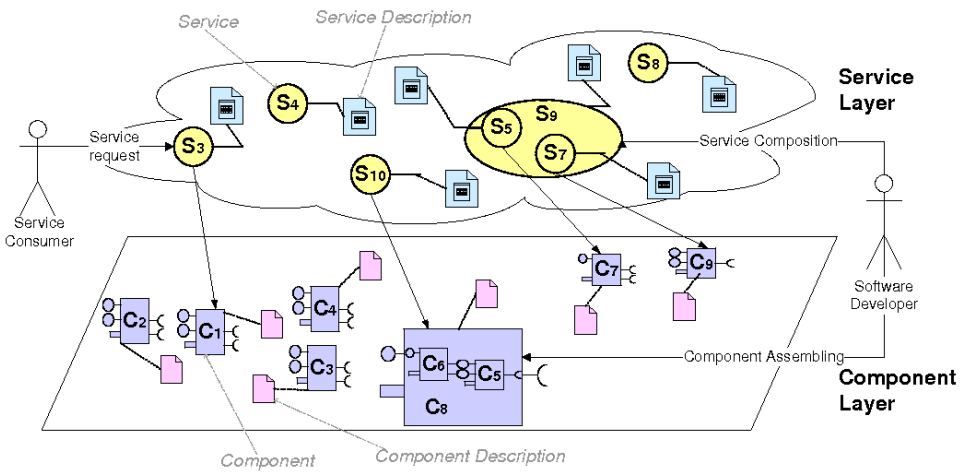


Fig. 2. Two-layer approach

operativity of the device.

Components developers bind to services by only using service descriptions. In other words, components contractually implement service descriptions but the selection of the specific component to instantiate is deferred at run-time by taking into account the service requester needs.

The service layer manages two aspects: (i) the description of services and their composition, (ii) the mapping between services and set of components implementing them.

Two main actors are represented in figure 2: Service Consumer and Software Developer. The former can only act at the service layer by formulating a service request. The request may refer to an existing service or may bring to assemble several services to provide a new service as specified from the Service Consumer. Software Developer may act either at the service layer by composing services, or at the component layer by implementing new services through component assembling.

In this concern we are extending our approach to the mediation of components interaction at the component layer [13, 15] (for concrete structural composition) to the service layer (for abstract composition). Inheriting and extending features of our approach that allows the synthesis of decentralized adaptor models, our intent is to automatically and dynamically synthesize behavioral models for guiding the application while performing its work-flow between services. In this way we address the adaptable composition of services, at the component and service level, in a way that guarantees correctness of the composition vs functional properties.

However, the services should also be adaptable according to the context-of-use, the user expected QoS and related SLA. By extending our framework for the development and deployment of adaptable component-based applications [8], our attempt is to offer a methodology that also assists the development and deployment of services with respect to the various dimensions of resource awareness.

5 The PLASTIC conceptual model

In contexts such as the one of PLASTIC, where the focus is on service design and development, it is important for service designers and developers to agree on concepts and principles of SOA, so that they can have the best benefits, in terms of integration and adaptation, when using related technologies.

Services in PLASTIC are meant to be *adaptable* to the context whether related to the user setting (e.g. type of device, user mobility) or the available network resources (e.g. network bandwidth). The goal is to provide services that achieve the best tradeoff between the user needs (in terms of Quality of Service) and the current context characteristics (in terms of available resources).

In order to achieve a high degree of adaptability, services must be aware of the context where they run. Thus, we make the context characteristics emerging at the service level: a service description will be not limited to its functional characteristics (i.e. signature and behaviour), but it will bring additional attributes that model its QoS characteristics (e.g. service reliability, latency time). Obviously, these attributes are parametric in that they may depend on the context.

Our approach is somehow opposite to a typical layered architectural model, where differences in a lower layer (in terms of functional and non-functional characteristics) disappear in the upper layer. In fact, the latter one aims at providing an uniform interface that hides the characteristics of the upper layer.

We believe that the heterogeneity of the current networks has to be exploited at the service level instead of being standardized through middlewares that aim at hiding differences among devices and connections. QoS becomes a tool in the hands of service architects and developers that can exploit this additional information to achieve the maximum user satisfaction.

In order to achieve this goal two main supports have to be provided: (i) non-functional models that link QoS at the platform/network level with QoS at the service level, (ii) Service Level Agreement strategies to achieve the best tradeoff.

Based on the considerations made above, we present the PLASTIC conceptual model, which is reported in Figure 3. In this figure we used two different graphical notation: rectangles to represent Plastic entities and humans to represent actors. The actors are entities external to the Plastic platform that interact with it in several ways. The modelled actor are: Service Developer, Service Provider, Service Consumer, Service Integrator, Service Registry and Component Assembler.

The first class entities in the conceptual model are *Software Service*, *Component Based Software System* and *Context*. The software functionalities that a system provides correspond to software services that are implemented by means of component based software systems. The software services might be combined to obtain complex services. The components and services compositions take into account the characteristics of the context the software services will be hosted in.

More in details, a software service is developed by a *Service Developer* and is a *Provided Functionality* that is, in its turn, a logical *Resource*. Whenever a software service is implemented, the *Service Provider* (that can coincide with the

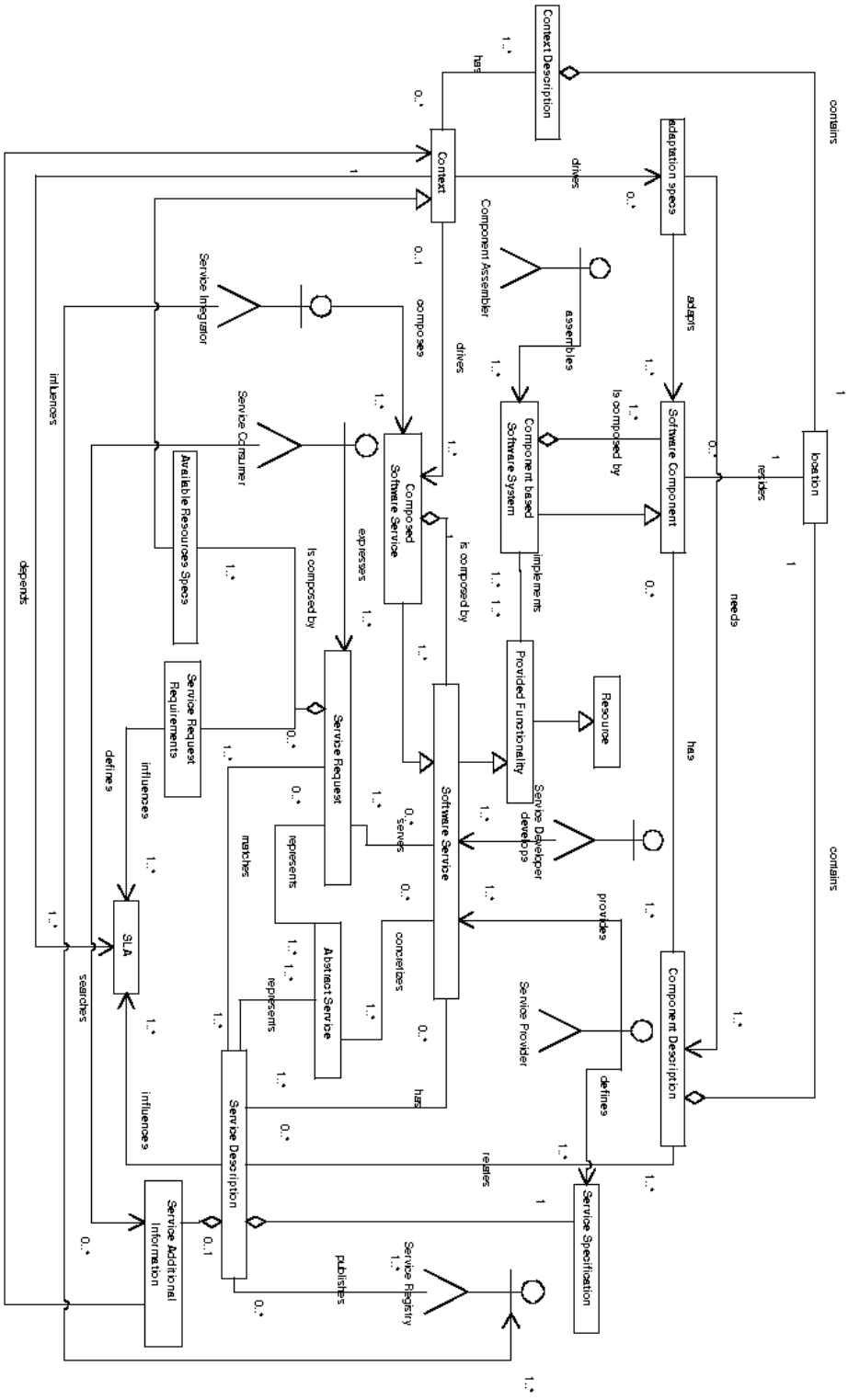


Fig. 3. Conceptual Model of Plastic vision

service developer) provides it for future usage. The service provider is the network-addressable entity that accepts and executes requests from *Service consumers*. To

make services accessible from the consumers it publishes them in a *Service Registry*.

A service registry is a network-based directory that contains available services. It is an entity that accepts and stores service description from service providers and provides those description to interested service consumers.

Software services can be simple or can result from composition of other software services, and in this case we name them as *Composed Software Services*. The *Service Integrator* is responsible of such composition to obtain more complex services required by service consumers. To deal with service composition, the integrator has to retrieve the software services and related information by involving the *Service Registry*.

A *Service Description* is associated to each service and it is composed by: (i) a *Service Specification* defined by the service provider and describing functional and non functional characteristics of the provided service; (ii) optionally a *Service Additional Information* that represents the feedback from the service customers that used the service.

According to the SecSE conceptual model, Service specification contains many information about the service: the behavior of the service, the service signature, other services required for its function, optionally its operational semantics and the service invariant, the specification of the exception thrown from the service. Again it contains non functional information: its price, the service policy, QoS such as security, dependability and performance, its compatibility with existing standard, and so on. Service Additional Information contains other information of the service specified, this time, from the users. Such information are collected and stored by the user during and after the execution of the service. it contains usage history, the satisfaction degree of the service, information related to the execution environments, the profile of the typical user of the service and the typical access devices used.

The service consumer asks for a service by expressing a *Service Request*. The service request hence specifies several *Service Request Requirements* that cover functional and non functional aspects. When a suitable service is found in the registry or a new one is implemented, the corresponding service description matches the request. The service request also might bring information on the *Available Resources* at the consumer side such as the available memory, the size and the characteristics of the screen, the type of network connectivity the device is using to connect itself to the Plastic platform. Such information is a piece of information of the *Context* where the service will run and will be used by the Plastic platform to adapt the required service to the device features.

According to the SecSE conceptual model, the service request as well as the service description are representations of an *Abstract Service* that can be concretized or not by a software service.

Each provided functionality, hence each service, is implemented by a *Component based Software System* that is composed by a set of *Software Components* assembled by the *Component Assembler*. The component assembler identifies the

software components to use. In particular, such components can be COTS, newly implemented ones or another Component based Software System (since it is recursively a software component). The software component has associated a *Component Description* specifying functional and non functional aspects of the component itself. The non functional aspects, in particular QoS, of a component are related to the context where the service will be provided. This information is provided by means of the *location* entity.

Since service characteristics, especially QoS aspects, are affected by the component quality, in the model there is a relation between service description and component description to represent this dependency.

The context is a relevant concept in our vision and it represents the logical and physical resources available in the service provision. The context has a *Context Description* that contains the description of the resources in terms of hardware devices, network connectivity and software services available in the execution environment. We can use context description in the adaptation of software components, in software services composition and in the *SLA* procedure when a new service request is formulated.

In a context-aware and adaptable system domain where services have to respect the QoS requests made by the consumer, the context drives the component adaptation. Adaptation is made at the software component level and combine information of the context, SLA and component description. In the conceptual model, *adaptation specs* is the entity that contains the specification of the adaptation rules. The definition of such rules is driven by the execution context.

Since we have also to model mobility, the context description contains an explicit entity for the *location*. Location is an entity that identifies the execution environment of a component, this implies that each context has a location identification and component description contains a reference to the location where it will be performed.

Finally the SLA is an entity modeling the conditions accepted by both the service consumer and the service provider. It represents a kind of contract that is obviously influenced by the service request requirements, the service description and the context where the service had to be provided. When a new service request is formulates, the PLASTIC platform has to contract the QoS of the service on the basis of the service request, the context the service has to be provided and the service descriptions of similar services already available by some providers. The contractual procedure may terminate with an agreement about QoS of the service from the consumer and the provider, or with no agreement. In case of agreement, an SLA is reached.

6 An example scenario: video-streaming service for e-learning

In this section we depict a simple scenario relating elements involved in the scenario to instantiations of a concepts in the conceptual model. Firstly, the scenario is described from the service consumer side and, secondly, from the PLASTIC

platform side.

6.1 Video-streaming service: from the consumer side

Alice has subscribed to PLASTIC platform and she has deployed on its mobile phone an application that allows her for accessing PLASTIC services.

Alice wants to access a short e-learning video session on marketing strategies from her mobile phone while she travels to get to the place where she has an interview for a job. Since Alice does not want to wait for the complete download of the video file before being able to watch it, she expresses the service request for the video via "video streaming". Moreover she wants to spend at most 25 euros, to have an high video quality and a medium audio quality, and to pay by credit card.

After having processed the Alice's request, the PLASTIC platform provides her with a list of video streaming hosting service providers (Stream IT, WebStream, Stream USA, StreamCast, StreamIsland, Web Streaming Hosting, etc) whose service characteristics almost satisfy the requirements Alice expressed in her request. For each provider it is also showed a table reporting some metrics of the trade-off level reached between service characteristics and Alice's request requirements about cost, video and audio quality, and payment method. Alice chooses StreamIsland since the reached compromise is suitable for her even though the video quality is not the wanted one. She also specifies that, whenever a better video quality will be available later, she wants to experience that quality.

To initiate the streaming, Alice has to provide credit card number, company, and expiration date. Within few seconds, a message acknowledges that the credit card transaction has been successfully completed and the showing of the e-learning session starts. After only five minutes Alice notes, with satisfaction, a sensible improvement of the video quality with more brightness and more color definition.

At the end of the video Alice decide to send a positive feedback.

6.2 Video-streaming service: from the provider side

After that the Alice's *service request* has been received, the PLASTIC platform searches the available *service registries* to find *service providers* for particular *service descriptions* that match the Alice's request. The request handler takes into account both (i) the *available resources specifications* of the Alice's mobile phone (i.e., memory, screen resolution, processor power that represent the consumer side execution context) and (ii) the *service request requirements* expressed by Alice (i.e., desired cost, video quality, and payment method)

Since more then one providers have published their streaming service descriptions into the service registry, the search activity successfully returns the reference list of the available services and relative providers (i.e., Stream IT, WebStream, Stream USA, StreamCast, StreamIsland, Web Streaming Hosting, etc)). All the providers are able to supply the e-learning video session respecting the available resources of the Alice's mobile phone so *adaptation* to the execution context can be

fulfilled. Unfortunately, there is no providers that is able to fully satisfy the requirements of Alice so negotiation is necessary. The negotiation phase starts by showing the table reporting the reached trade-off between service characteristics (for each provider) and service request requirements. In our case this phase quickly finish. In fact, even though Alice's request requirements are not completely satisfied and she would appreciate a better service if it is available later, she chooses a provider and hence there no need to reiterate the negotiation. In other word the *SLA*, representing the contract between the service consumer and the service provider, has been established.

After that the provider StreamIsland has been selected, it returns a reference to a component implementing the streaming functionality that has been set according to the request requirements and that has been adapted to the consumer side context. Actually, the service required by Alice represent a *composed software service* capable of combining and dynamically discovering other services in order to accomplish the task. In particular the overall required service is implemented as the *composition* of at least the component implementing the payment service and the component offering the streaming. Moreover, new arrival service provider capable of offering a better video quality (to completely satisfy the Alice's request) have to be discovered.

That is, Alice notes the improvement of the video quality because a new arrival provider has been discovered. It disposes of a new component that implements the streaming functionality and that is capable of obtaining a better video quality. By buffering the streaming channel, the old component is dynamically adapted and replaced with the old one. Alice only see the improvement. At the end of the e-learning video session the *additional information* defined by Alice (by means of the sent feedbacks) are added as part of the service description.

7 Related Work

Recently, several approaches to conceptualize the world of services have been proposed.

Our model is inspired by the SeCSE conceptual model [12]. This model aims at providing a common terminology across the SeCSE project [18] for letting all the involved people communicate with each other using a common language and a common understanding.

The model has been conceived to be extensible since it can easily adapted in order to accommodate different needs for different service-oriented domain.

The SeCSE conceptual model was initially inspired by the Web Service Architecture [20] (WSA) drafted by the W3C. The SeCSE model can be seen as complementary to the one of WSA since SeCSE try to clarify and detail more than the WSA does for the concept of service.

The WSA conceptual model [20] is structured in four areas, each focusing on a specific aspect: (i) *Service Model* for service concept relationships; (ii) *Message Oriented Model* for message transmission, processing and the relationship between

message senders and receivers; (iii)*Resource Oriented Model* for uniquely identified, owned resources that are relevant to Web services; (iv) *Policy Model* for modelling aspects of the architecture that relate to constraints, security and QoS.

The SeCSE model marginally considers the message oriented, resource oriented, and policy models of the WSA, but it tries to clarify and detail more than the WSA does for the concept of service, and service-related activities (i.e., publication, discovery, composition, and monitoring).

Also, the WSA message models are not concerned with any semantic significance of the content of a message or relationships to other messages; the SeCSE conceptual model makes clear the relationships between service description, semantics, and service interface.

The work in [14] proposes a SOA reference model for identifying the entities involved in a service-oriented environment, for understanding the roles played by them and for establishing relationships between them. The proposed reference model is a truly abstract framework that provide an higher level commonality by supply a large set of definitions that any SOA based initiative should take into account. The model is not tied to any particular implementation details, standards and technologies. Focussing on software architectures, it intends to be the basis for describing reference architectures and patterns that can be used to define particular SOA.

8 Conclusions

We have introduced an initial conceptual model for adaptable context-aware service-oriented applications. The conceptual model can be considered as a dictionary for users and developers in this domain. The main original aspects of our work can be summarized as: lumping in the same conceptual model component-based and service-based concepts, and building relationships between these concepts; introducing the context concept that allow to define adaptable services; spreading over different concepts QoS issues, thus allowing services to provide the best QoS characteristics (through SLA), as a tradeoff between offered QoS (from the network) and required QoS (from the users). The QoS attributes migrate at the application level, so they become a tool in the hand of developers/integrators to achieve the best customer satisfaction.

The conceptual model presented in this paper needs to be refined and extended in two main directions. On one hand, the context concept shall be detailed and related to other concepts, on the other hand the QoS attributes shall be made explicit across the other model concepts.

References

[1] Special Issue on Applications and Services for the B3G/4G Era, October 2006.

[2] A. Bertolino and W. Emmerich and P. Inverardi and V. Issarny. Softure: Adaptable,

Reliable and Performing Software for the Future. *Future Research Challenges for Software and Services (FRCSS)*, 2006.

[3] A. M. Sassen and C. Macmillan. The service engineering area: An overview of its current state and a vision of its future. Technical report, Euorepean Commission, July 2005.

[4] AA.VV. Proceedings of sigsoft component-based software system symposium, 1998-2006.

[5] Z. Baida, J. Gordijn, B. Omelayenko, and H. Akkermans. A shared service terminology for online service provisioning. In *Proceedings of the 6th international conference on Electronic commerce, p.1-10*, 2004.

[6] D. Sprott and L. Wilkes. Understanding Service-Oriented Architecture. http://msdn.microsoft.com/architecture/soa/, Microsoft, 2004.

[7] E. Christensen and F. Curbera and G. Meredith and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.

[8] F.Mancinelli and P. Inverardi. A resource model for adaptable applications. In *To apper, Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), (ICSE)*, May 2006.

[9] H. Cervantes and R. S. Hall. Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. In *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, Edinburgh, Scotland, May 2004.

[10] IBM. WebSphere MQ. http://www.ibm.com/software/mqseries/.

[11] IBM. Migration to a service-oriented architecture Part1. http://www-128.ibm.com/developerworks/library/ws-migratesoa, 2003.

[12] M. Colombo and E. Di Nitto and M. Di Penta and D. Distante and Maurilio Zuccalà. Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems. In *3rd International Conference on Service Oriented Computing (ICSOC)*, Amsterdam, the Netherlands, December 2005.

[13] M.Tivoli and M.Autili. Synthesis: a tool for synthesizing "correct" and protocol-enhanced adaptors. *RSTI, L'OBJET JOURNAL 12/2006, pages 77 to 103*, 2006.

[14] OASIS. Reference Model for Service Oriented Architecture. Committee Draft 1.0, 7, 2006.

[15] P.Inverardi, L.Mostarda, M.Tivoli, and M.Autili. Synthesis of correct and distributed adaptors for component-based systems: an automatic approach. In *Proceedings of Automated Software Engineering (ASE)*, 2005.

[16] PLASTIC Project. PLASTIC Website. http://www.ist-plastic.org, 2005.

[17] Chris Preist. A conceptual architecture for semantic web services. In *Proceedings of the International Semantic Web Conference (ISWC)*, November 2004.

[18] SeCSE Project. SeCSE Website. http://secse.eng.it, 2004.

[19] SUN. Service Oriented Architecture. http://www.theserverside.com, 2003.

[20] W3C Working Group. Web Services Architecture (WSA). http://www.w3.org/TR/ws-arch/, February 2004.

Modeling of Reliable Messaging in Service Oriented Architectures [3]

László Gönczy [1] and Dániel Varró [2]

Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary

Abstract

Due to the increasing need of highly dependable services in Service-Oriented Architectures (SOA), service-level agreements include more and more frequently such traditional aspects as security, safety, availability, reliability, etc. Whenever a service can no longer be provided with the required QoS, the service requester need to switch dynamically to a new service having adequate service parameters. In the current paper, we propose a metamodel to capture such parameters required for reliable messaging in services in a semi-formal way (as an extension to [1]). Furthermore, we incorporate fault-tolerant algorithms into appropriate reconfiguration mechanisms for modeling reliable message delivery by graph transformation rules.

Key words: Service Oriented Architecture, Graph Transformation, Reliable Messaging,

[1] Email: gonczy@mit.bme.hu
[2] Email: varro@mit.bme.hu
[3] This work was partially supported by the SENSORIA European project (IST-3-016004).

1 Introduction

Service-Oriented Architectures (SOA) provide a flexible and dynamic platform for implementing business services. The main business-level driver of the SOA paradigm is *componentization*, which raises the level of abstraction from objects to services in the design process of distributed applications. The main architectural-level driver of the SOA paradigm is to provide a common middleware framework for dynamic discovery, interaction and reconfiguration of service components independently of the actual business environment.

Due to the increasing need of highly dependable services, service-level agreements include more and more frequently such traditional (non-functional) aspects as security, safety, availability, reliability, etc. The general idea is that whenever a service can no longer be provided with the required QoS, the service requester needs to switch dynamically to a new service having adequate service parameters.

In an ideal scenario of using service-level agreements, designers only specify the requirements for a specific service, and reconfigurations aiming to maintain the required QoS parameters are handled automatically by the underlying service middleware platform. Therefore, the service requestor does not need to be adapted explicitly (i.e. on the code level) to the evolution of the environment.

Recently, the identification of non-functional parameters of services have been addressed by various XML-based standards related to web services (such as WS-Reliable Messaging, WS-Reliable Messaging Policies, etc.). A focal topic in many of these standards is related to reliable messaging between services, where the delivery of a message can be guaranteed by the underlying platform by appropriate reconfiguration mechanisms. In contrast to the *specification* of these reliability service properties, currently only very experimental solutions exist in the industry (such as RAMP-Toolkit [13] by IBM or RM4GS [14] by a consortium led by Fujitsu-Siemens, Hitachi and NEC) that actually implement these reconfigurations in order to maintain the required level of reliability.

In the paper, we facilitate the use of a precise model-based approach for the development of high-level, reconfiguration mechanisms required for reliable messaging in the underlying service middleware. Our long term goal is automatically derive implementations of reliable messaging on various existing platforms based directly upon provenly correct dynamic reconfiguration mechanisms.

In the current paper, we conceptually follow [1] where a semi-formal platform-independent and a SOA-specific metamodel (ontology) was developed to capture service architectures on various levels of abstraction in a model-driven service development process. Furthermore, reconfigurations for service publishing, querying and binding were captured by graph transformation rules [4], which provide a formal, rule and pattern-based specification formalism widely used in various application areas. This combination of metamodeling and graph transformation rules fits well to a model-based development process for service middleware.

This paper extends the core metamodel defined in [1] (and overviewed in Sec.3) by a new package for reliable messaging (Sec. 4.2). Moreover, we provide new

reconfiguration primitives for reliable message delivery in the form of graph transformation rules (Sec. 5.2) by integrating dependability techniques [10].

2 Service Reconfigurations for Reliable Messaging: An Overview

Our overall research objectives towards provenly correct service reconfigurations for reliable messaging follows the SENSORIA approach [16], and it is sketched in Fig. 1.

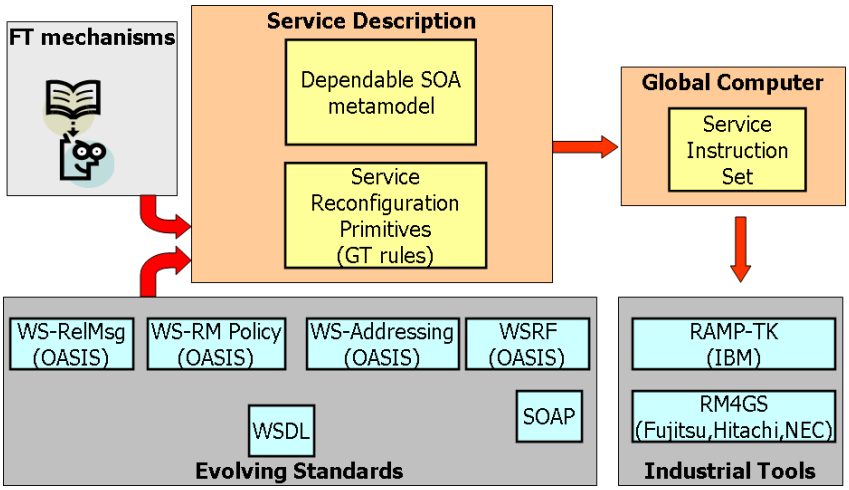


Fig. 1. Towards Provenly Correct Service Reconfigurations for Reliable Messaging

- A *dependability extension of the core SOA metamodel* will be developed which is synthesized from existing standards related to web services and other application areas. The current paper provides a first step towards this by focusing on service parameters for reliable messaging.
- *High-level reconfiguration primitives* will be defined to capture dependable services in the form of graph transformation rules by integrating traditional fault-tolerance techniques into SOA. In the current paper, basic reconfiguration steps are identified for providing reliable messaging between services.
- *Formal analysis* will be carried out in order to justify the correctness of reconfigurations.
- A new kind of virtual machine (called a *global computer*) is envisaged with a special *instruction set* tailored to dependable service reconfigurations.
- Finally, this instruction set will be *mapped to existing technologies* dedicated to the development of reliable web services (such as RAMP [13] or RM4GS [14])

3 Core SOA Metamodel

To illustrate the problem domain with a small but practical example, the case study of a *course management system* is introduced.

Let us consider a university where the students can perform their administrative tasks (for instance, signing for courses and exams) online. Hereby we model the *exam management service* within this course management system. The exam management component offers services to students, teachers and other employees of the university like the administrative staff. Students can sign up for or cancel an exam while teachers can submit the result of the exams. Administrative staff can perform queries against the exam database (for instance, to retrieve the course with the highest failure ratio).

The main architectural concepts of the domain of service-oriented architectures are captured by a corresponding metamodel. The metamodel of "core" SOA functionality is shown in Fig. 2. It is based on the metamodel presented in [1], with a minor modification of merging both the structural and dynamic aspects into a single package.

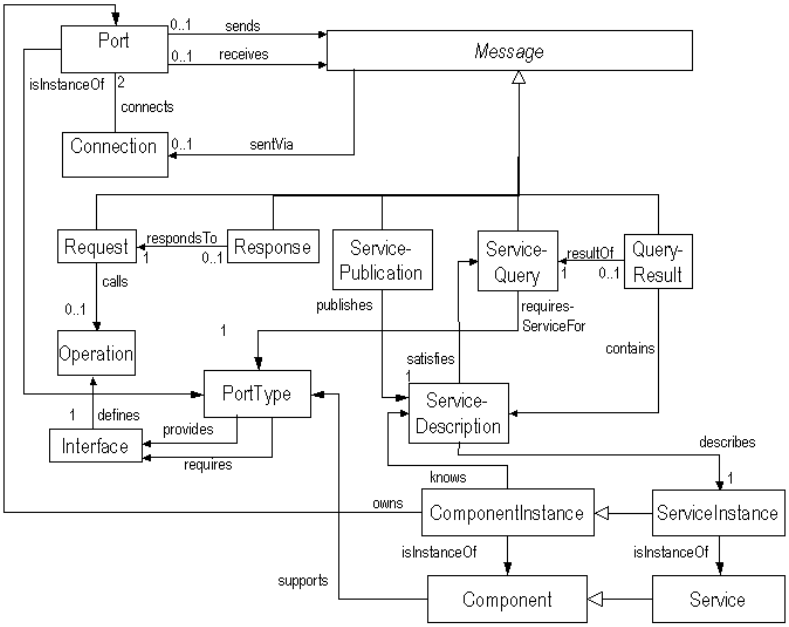


Fig. 2. Core metamodel of SOA

The core model to service-oriented architectures consists of the following elements:

- A *component* is a basic "module" in the system which provides a service. In the online course management system, for instance, ExamAdministration, ExamRegistration , ExamQuery will be such components.
- A *service* is a set of well-defined functionality. In our case, there will be three services: ExamAdministrationService for educational staff to create new exams, set the parameters of exams (such as date, limit, etc.) and to upload the

results of the exams; ExamRegistrationService for students to register themselves to exams (and to cancel registrations, if necessary) and ExamQuery for the administrative staff to retrieve statistics on exams.These will be provided by the three components, respectively.

- A *port* is the communication ”endpoint” of the service, with a set of abstract operations and messages. For instance, ExamRegistrationPort is the interface of ExamRegistrationService.
- A *connection* denotes a bidirectional channel between two ports at run-time.
- An *operation* is an ”atomic” action with input and output messages. There can be multiple operations defined on the same port. SignUp is an operation, defined on ExamRegistrationPort.
- A *message* is a set of parameters with pre-defined types. For instance, ExamDescription message may have parameters like studentName, examId, etc. The abstract message class is refined into the following subtypes of messages: *response, request, service publication, query and query result*.
- A *service description* is a descriptor file containing all necessary information about the runtime cooperation with the service, such as description of port, operations, messages, etc.

4 Extensions for Reliable Messaging in Web Services

4.1 Non-functional Requirements in Existing Web Service Technologies

Although there exist some initiatives to define the so-called "non-functional" properties of services, such as Web Services Modeling Ontology [18], W3C Web Services Architecture [17], DublinCore Metadata for ServiceDiscovery [3], the terminology is still ambiguous.

To illustrate the modeling of non-functional properties by a practical and simple example, hereby we present a model-based reconfiguration for reliable messaging to tolerate communication faults. As the consumers of the Web services are not aware of the details of underlying network protocol, the semantics of the message delivery has to be specified at the application level as requirements for reliable messaging. This needs a platform-independent representation of message attributes, which is reflected by a number of emerging standards, such as [20] and [19], which are converging to each other [21]. Some reference implementations for popular application servers like IBM WebSphere or Apache Tomcat are available.

These industrial standards and initiatives usually suppose that the service provide signs a contract with each client about the Quality of Service, measured in terms such as average response time, minimal throughput, type of message delivery, etc. These contracts are typically identical for classes of similar clients (roles), for instance, Golden User, Business Partner, Individual Customer, etc. The runtime service instances send their messages according to these contracts, and the additional information, regarding these non-functional aspects, is hidden from the

application layer, so that the modification of the original clients on the consumers' side is not necessary. The additional information is handled by components aware of reliability attributes, called *"Reliable Message Endpoints"*. Technologically speaking, the header of SOAP envelopes is extended with some attributes by a "Reliable Message Endpoint" on the provider's side, which are then removed from the messages by another "Reliable Message Endpoint" at the client side. Since the concrete format of these attributes in message headers is out of our scope, here we model an abstract description, which, however, will be mapped to existing technologies using model transformation techniques in the future.

4.2 Metamodel extensions for reliable messaging in services

Now we extend the core SOA metamodel of [1] to capture properties of reliable messaging between services. After enriching the domain metamodel, our long term goal is to define a corresponding UML profile to provide extensions to the UML language tailored to a specific application domain by introducing domain concepts, attributes and relations in the form of stereotypes and tagged values. However, the current paper only focuses on designing extensions for reliable messaging in the SOA metamodel.

In the current paper, we first derive a subclass from SOA element in the reliable SOA metamodel, and then create an association from the child class (e.g. RelMsgEnvelope) to the parent class (e.g. Message) in addition. As a result, original SOA reconfiguration rules defined in [1] are still applicable with the extensions for the reliable messaging metamodel, thus we can handled messaging between services on the communication level in the same way as before. Furthermore, the original messages are kept but wrapped into an envelope by introducing a new association.

The extensions of the SOA metamodel for reliable messaging is presented in Fig. 3:

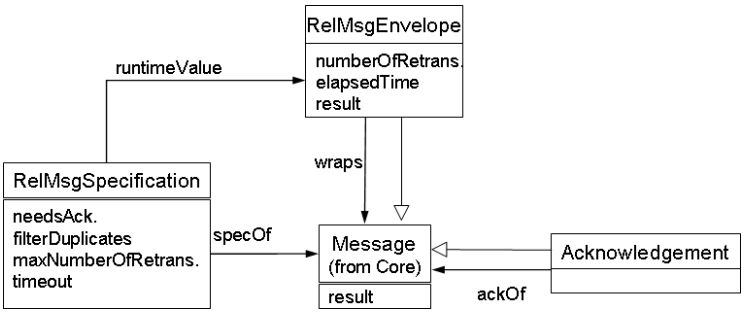


Fig. 3. Metamodel of Reliability Extensions

- RelMsgSpecification is a class for specifying the requirements for reliable messaging between SOA services (see association specOf).
  - Attribute needsAck is a boolean value to express if an acknowledgement should be sent to a message. If an acknowledgement arrives to the sender for a message, then it is guaranteed that the message is sent at least once.

- Attribute filterDuplicates is a boolean value to express that a message should be accepted and processed by the receiver at most once.
- Attribute timeout is a timer constraint which specifies how much does the sender waits for the acknowledgement of a message before retransmission.
- Attribute maxNumberOfRetrans is an integer which puts an upper limit how many times a message can be retransmitted by the sender due to the lack of acknowledgement from the receiver.

- RelMsgEnvelope is a subclass of core SOA Message which serves as an envelope for wrapping up the real message to be sent (wraps).
  - Attribute numberOfRetrans is a serial number for the envelope which is increased by one each time the same message is retransmitted.
  - Attribute timeElapsed denotes the time elapsed since the (last) transmission of a message.
- Acknowledgement is a subclass of core SOA Message which denotes an acknowledgement sent in response to a message.

As this extension reflects to existing standards, implementations of models can be derived following the Model Driven Architecture (MDA, [12]) approach. Runtime values of attributes will be generated from XML descriptors. The automated generation of XML descriptors from high level models is part of our future research.

In the course management application, let us consider the scenario of signing up for an exam. The two communicating components are ClientApp and ExamRegistration. The client signs up for a particular exam by sending a message to the instance port of ExamRegistrationProviderPT port type. This operation needs an *ExactlyOnce* messaging semantics, denoted by the attributes of the reliability specification of the message. Fig. 4 shows the relevant part of the instance graph of the system.

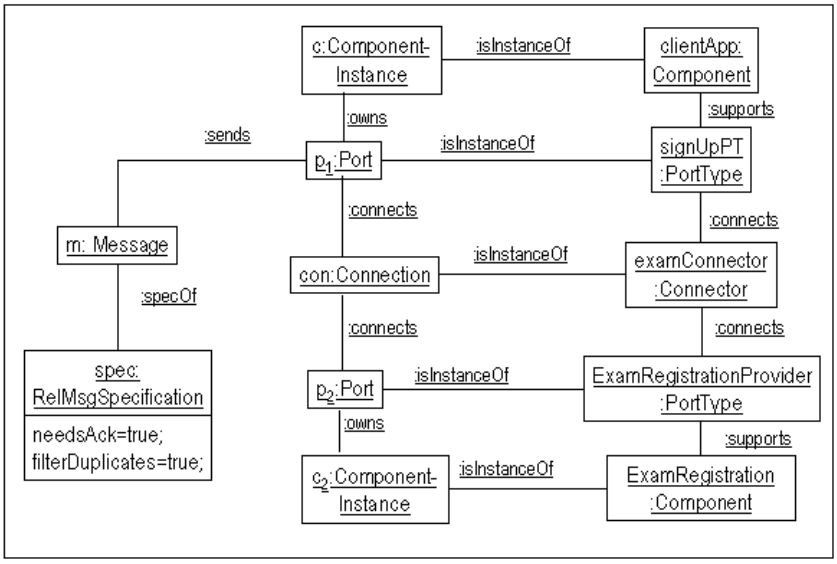


Fig. 4. Instance graph of the exam management system

Message sending operation will be executed by the application of transformations on the instance graph as described in Sect. 5.

5 Reconfiguration for Reliable SOA Messaging by Graph Transformation

We now propose to describe the reconfiguration mechanisms of reliable SOA messaging by graph transformation rules (conceptually following the approach presented in [1]).

5.1 Overview of graph transformation

A main benefit of using graph transformations as a formal specification paradigm for capturing reconfiguration rules is that they are visual, intuitive, therefore they can be understood by service engineers as well. The interested reader may find a detailed theoretical discussion of graph transformation in [4], here we present just a brief overview on it.

Furthermore, graph transformation allows dynamic metamodeling [7] in a certain domain. The high-level (ontological) concepts are visualized as UML class diagrams while graph patterns are considered to be UML object diagrams to express that concrete models are instances (objects) of the metamodel (classes). This combines the advantage of precise modeling and visual design, as the ontology "behind" the class diagram defines the semantics of the model, while the graph transformation rules, interpreted on concrete model instances, can be designed visually by existing graph transformation tools.

A graph transformation rule consists of a *Left Hand Side (LHS)*, a *Right Hand Side (RHS)* and optionally a *Negative Application Condition (NAC)*. The *LHS* is a graph pattern consisting of the *mandatory* elements which prescribes a precondition for the application of the rule. The *RHS* is a graph pattern containing all elements which should be present after the application of the rule. Elements in the $RHS \cap LHS$ are unchanged after the execution of the transformation, elements in $LHS \setminus RHS$ are deleted while elements in $RHS \setminus LHS$ are newly created by the rule. The fulfillment of the negative condition prevents the rule from being executed on the particular matching. Furthermore, graph transformation rules frequently allow the use of *attribute conditions (constraints)* which restrict attributes of the matched nodes, and *attribute assignments*, which may describe the updates of certain attributes as a result of rule application. Hereby we follow the Double Pushout (DPO) approach [4] for the semantics of graph transformation.

A *Graph Transformation System, GTS* consists of a graph instance and the transformation rules. The execution of a GTS is nondeterministic, since the next rule to be applied and the matching on which a rule is applied is not restricted (by default) by additional control information.

In this paper, we use a compact visualization of graph transformation rules (first introduced in the Fujaba framework [6]), when the entire rule is merged into a

single pattern. Newly created elements are denoted by *{new}* and by slashed lines while deleted elements have a grey background and a *{deleted}* tag. Elements in the subset of the *LHS* and the *RHS* are visualized normally, and elements of NAC are crossed out. In the current paper, the *{new}* tag implicitly implicates a negative condition as well, which prevents the rule from creating infinite number of new elements on the same matching (in the case of messaging, the same message is received only once).

Rules of Fig. 5 (which is a simplified version of rules presented in [1]) illustrate message sending and receiving in Service Oriented Architectures with the "traditional" and the compact visualization style.

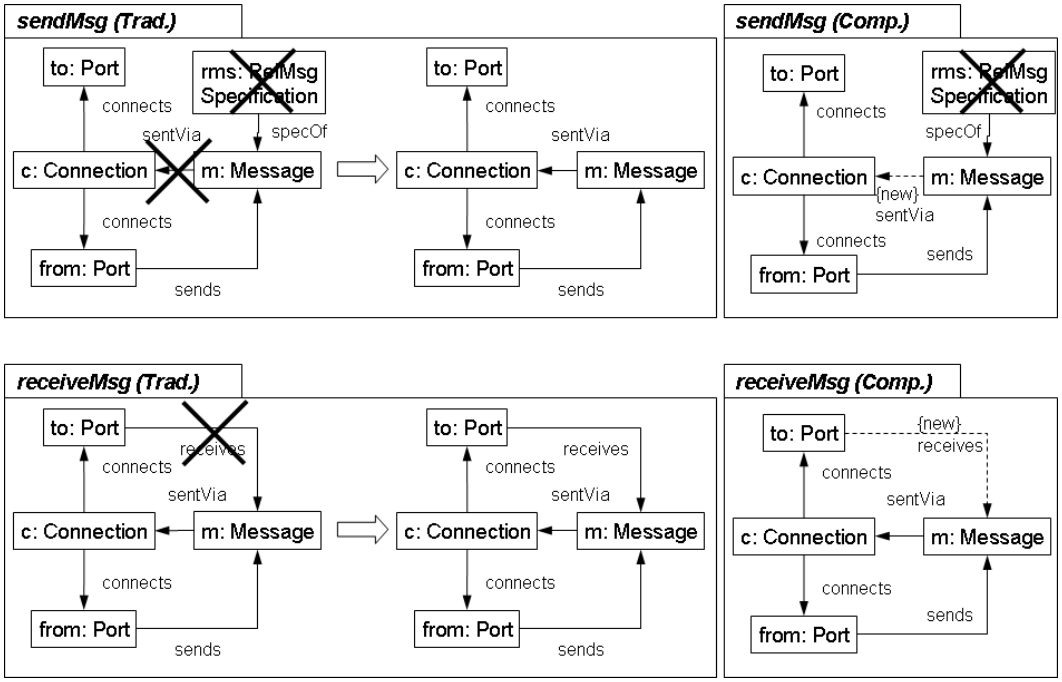


Fig. 5. Sending and receiving a message

The rules in Fig. 5 capture the operations of the basic messaging. Two services can communicate if there is an open connection between them. The message is first sent via the connection (a new sentVia association is created). The service provider can receive the message only once, this is implicated by the negative application condition of the receive rule.

5.2 Reconfiguration Rules

The reliable messaging can be assured by the following reconfiguration rules captured by graph transformation.

First, the normal messages have to be packed into and wrapped from envelopes (as in the case of present reliable messaging technologies). Thus, the messages are wrapped up in the sender side instead of being transmitted (Fig. 6) and envelopes are opened before receiving their content at the receiver side Fig. 7. As the type of

the message is of specified, these rules will match for instances of every subclass of message class with a reliability specification. Thus, reliable messaging is also provided for asynchronous service invocations, discovery queries, etc.

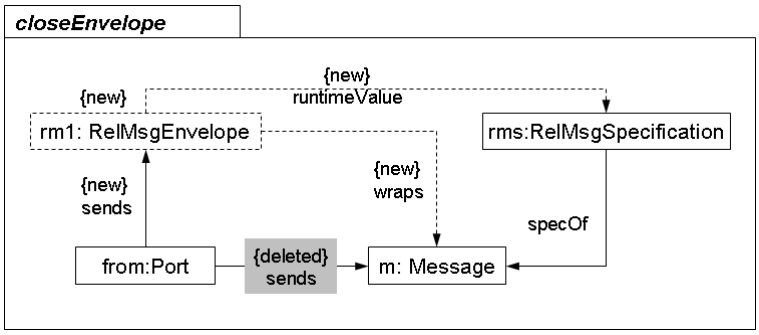


Fig. 6. Wrapping a Message into an Envelope

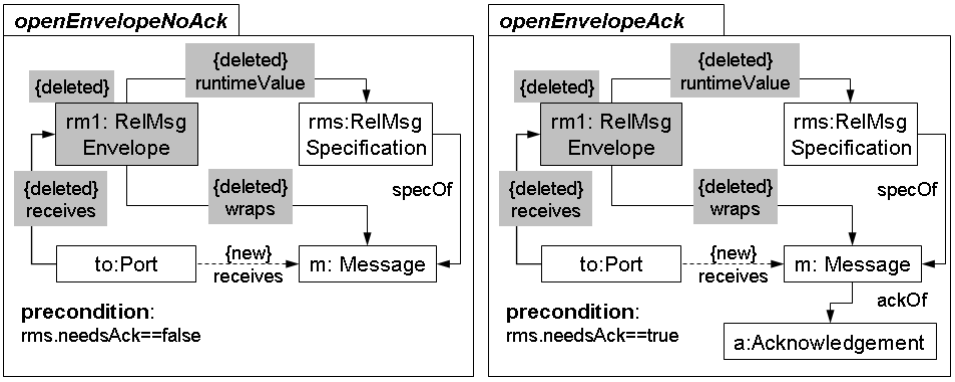


Fig. 7. Opening an Envelope

At the sender side, there are basically two message sending modes, depending on the value of the needsAck parameter of the ReliableMessageSpecification object describing the requirements for messaging.

At least once message delivery

If this parameter is true, reliable message sending required for a particular message, which corresponds to the *AtLeastOnce* messaging semantics. In this case, the sender will wait for an acknowledgement and consider the transmission of a message successful only if the acknowledgement arrives within the timeout interval. The rule of the successful message transmission (more precisely, the arrival of an acknowledgement in time) is shown in Fig. 8

On the other hand, if the acknowledgement does not arrive in time, then the next action (i.e. the next rule to be applied) depends on the number of retransmitted messages. If the retransmission number of a particular message is smaller than the allowed (precondition of rule *RetransmitMsg*), then a new instance of the ReliableMsgEnvelope class is created and sent with the same content and a higher retransmission number (rule *RetransmitMsg*). If the same message content cannot

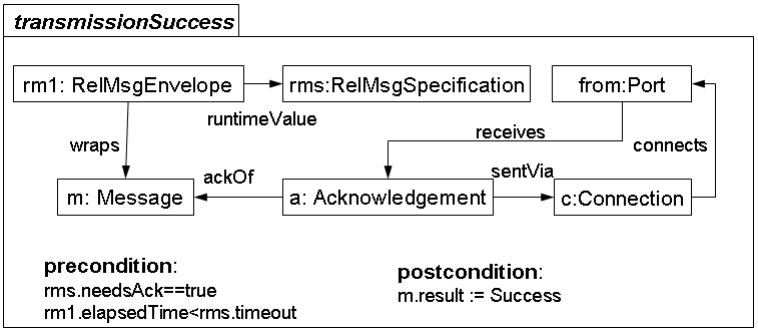


Fig. 8. Acknowledgement arrives in time

be sent again (precondition of rule *TransmissionFailure*), then the transmission of the message is considered to be failed. Note that if no acknowledgement is needed, then no additional rules are applied at message sending, only the core *"SendMsg"* rule matches the instance graph.

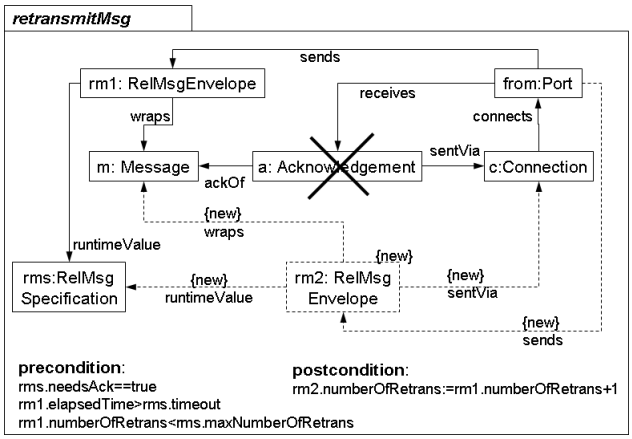


Fig. 9. Retransmission of a Message if Timeout Exceeded

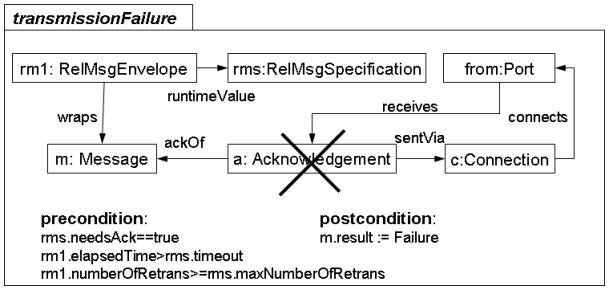


Fig. 10. Failure of Message Transmission

On the receiver side, the messages are acknowledged if needed (see Fig. 11), otherwise the core *ReceiveMsg* rule is applied.

Exactly once message delivery

Some applications may need the guarantee of the *ExactlyOnce* semantics, which prescribes acknowledgements and filtering (needsAck and filterDuplicates are set

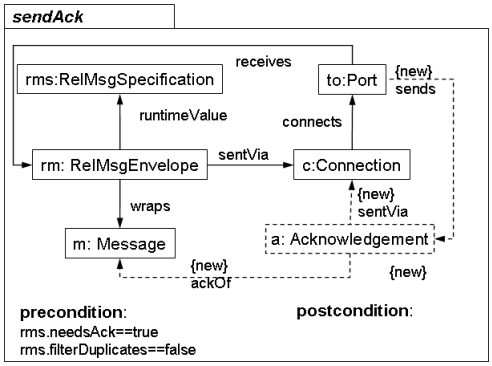


Fig. 11. Sending an Acknowledgement to an incoming message

to true, respectively).

At this communication mode, if the first instance arrives (no previous message has been received with the same content), then the message is received and an acknowledgement is created and sent back, as shown in Fig. 12.

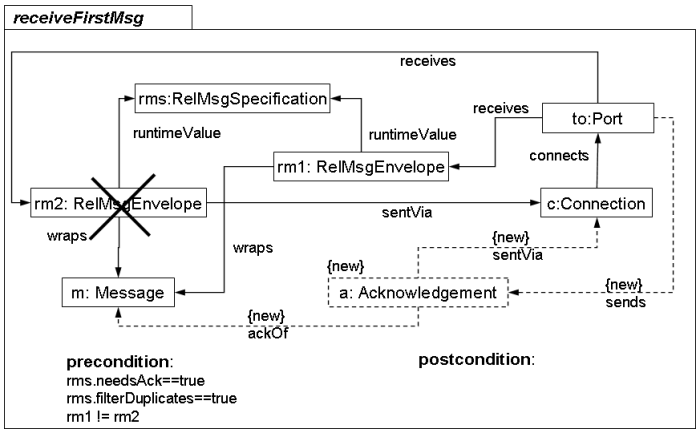


Fig. 12. Receiving the first message instance with the *ExactlyOnce* semantics

If the acknowledgement of a message is lost (due to some network error), then the sender will retransmit the same message after the timeout exceeds. In this case, a duplicate message will be received at the from Port. If the duplicates are filtered , then the arriving duplicate message is dropped but an acknowledgment is still created, as shown in Fig. 13.

In the case of registration for an exam, the successful registration may correspond to the following rule application sequence: *closeEnvelope*, *sendMsg*, *sendAck*, *openEnvelope*, *receiveMsg*, and *transmissionSuccess*.

6 Related work

Related works in this field usually concentrate either on describing the non-functional attributes of services, or on visual modeling of dynamic aspects of Service Oriented Architectures. Our work is based on the approach of [1]. [2] describes the application of graph transformations in the runtime matching of behavioral Web

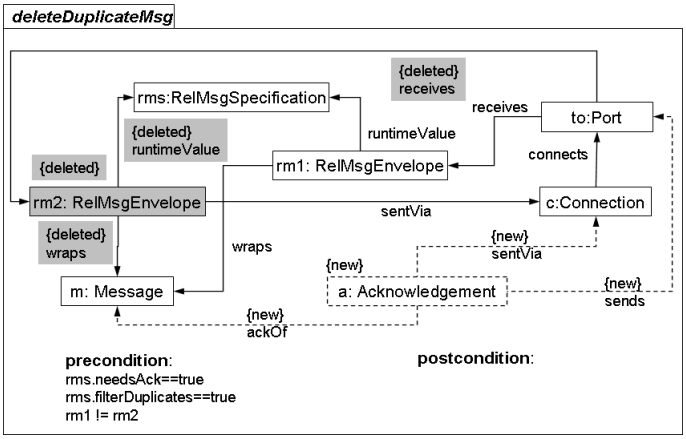


Fig. 13. Deleting duplicates

service specifications. In [8], the conformance testing of Web services is based on graph transformations, focusing on the automated test case generation. However, none of these works discusses the aspects of reliable messaging. Our aim was to utilize the benefits of this approach by extending the metamodel and the transformation rules.

Graph transformation is used as a specification technique for dynamic architectural reconfigurations in [5] using the algebraic framework CommUnity. Hirsch uses graph transformations over hypergraphs in [9] to specify run-time interactions among components, reconfigurations, and mobility in a given architectural style. However, the problem of reliable messaging in SOA is not addressed in either case.

A profile for reliability was designed for J2EE applications in [15]. Our work is different in the sense that we introduce a formal operational semantics of reliability messaging mechanisms which can provide basis for the underlying SOA middleware in an application independent way.

In [11], a pattern based specification and run-time validation approach is presented for interaction properties of web services using a semantic web rule language (SWRL). These patterns include constraints (requirements) on service invocations including *at-most-n* and *at-least-n* message delivery. This approach only reports run-time violation of the constraints, while our overall goal is to guarantee the delivery of messages with appropriate reliability semantics by the underlying middleware.

In the industrial field, there are existing and emerging specifications and technologies like [19], [20], [21]. However, their use is still ad-hoc and no model-based design-time support is available for reliable messaging.

7 Conclusion

In this paper we proposed an extension to the core SOA metamodel of [1] and a technique to capture the reconfiguration mechanisms to enhance the development of more robust SOA middleware. The main advantage of our solution is its seamless

integration with the previous initiative: core SOA reconfiguration mechanisms of [1] are directly applicable without changes, furthermore, the original messages are kept unaltered by the proposed wrap up mechanism.

We are currently working on the formal verification of the correctness of the proposed reconfiguration mechanisms using existing verification tools for graph transformation systems. As the next step in the future, we plan to implement the automatic generation of runtime implementation in existing middleware and to create test cases for reliable messaging.

References

[1] L.Baresi, R.Heckel, S.Thöne and D.Varró, *Style-Based Modeling and Refinement of Service-Oriented Architectures.* To appear in Journal of Software and Systems Modelling, 2006.

[2] A.Cherchago and R.Heckel. *Specification Matching of Web Services Using Conditional Graph Transformation Rules.* In Proc. of International Conference on Graph Transformations, 2004, LNCS Vol.**3256**, Springer, pp. 304-318.

[3] DublinCore Metadata Initiative, 2006. http://dublincore.org/

[4] H.Ehrig, M.Pfender and H.J.Schneider. "Graph grammars: an algebraic approach." 14th Annual IEEE Symposium on Switching and Automata Theory. pp. 167-180, IEEE, 1973.

[5] M.Wermelinger and J.L.Fiadeiro. *A graph transformation approach to software architecture reconfiguration.* Science of Computer Programming, 44(2):133Ű155, 2002.

[6] T.Fischer, J.Niere, L.Torunski and A.Zündorf, "Story Diagrams: A new Graph Transformation Language based on UML and Java", Proc. Theory and Application to Graph Transformations (TAGT'98), vol. 1764 of LNCS, 2000, Springer.

[7] J.H.Hausmann, R.Heckel, M.Lohmann: "Model-based Discovery of Web Services", In Proc. of the IEEE International Conference on Web Services (ICWS), June 6-9, 2004, USA,

[8] R.Heckel and L.Mariani. "Automated Conformance Testing of Web Services." In Proc. of 8th International Conference on Fundamental Approaches to Sofware Engineering (FASE 2005), vol. 3442 of LNCS, Springer, pp. 34-48.

[9] D.Hirsch. *Graph transformation models for software architecture styles*. PhD thesis, Departamento de Computacion, Universidad de Buenos Aires, 2003.

[10] J.Laprie, B.Randell, C.Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Transactions on Dependable and Secure Computing, (Vol.**1**, No.**1**) (2004) pp. 11-33

[11] Zheng Li, Jun Han, Yan Jin, "Pattern-Based Specification and Validation of Web Services Interaction Properties", In Proc. Third International Conference on Service-Oriented Computing (ICSOC 2005), Vol. 3826 of LNCS, Springer, pp. 73-86.

[12] Object Management Group. Model Driven Architecture. http://www.omg.org/mda

[13] Reliable Asynchronous Message Profile (RAMP) Toolkit, IBM alphaworks. http://www.alphaworks.ibm.com/tech/ramptk

[14] RM4GS Reference Guide, Version 1.0, FUJITSU LIMITED, Hitachi, Ltd. and NEC Corporation, 2004. http://xml.coverpages.org/rm4gs20041125-reference.pdf

[15] G.N.Rodrigues, G.Roberts, W.Emmerich and J.Skene. "Reliability Support for the Model Driven Architecture." In Proc. Workshop on Software Architectures for Dependable Systems (WADS 2003), Vol. 3069 of LNCS, Springer, pp. 79-98, 2004.

[16] Software Engineering for Service-Oriented Overlay Computers (SENSORIA) European project IST-3-016004. http://sensoria.fast.de

[17] Web Services Architecture, 2004. http://www.w3.org/TR/ws-arch/

[18] Web Service Modeling Ontology (WSMO), W3C Member Submission, 2005. http://www.w3.org/Submission/2005/SUBM-WSMO-20050603/

[19] Web Services Reliable Messaging Protocol (WS-ReliableMessaging) BEA Systems, IBM, Microsoft Corporation, Inc, and TIBCO Software Inc., 2002.

[20] Web Services Reliable Messaging TC WS-Reliability 1.1 Committee Draft 1.086, OASIS Open Consortium, 2004.

[21] Call for Participation in the OASIS Web Services Reliable Exchange (WS-RX), OASIS Open Consortium, 2005. http://xml.coverpages.org/ni2005-05-04-a.html

A Survey on Services Composition Languages and Models

Antonio Bucchiarone [1]

Istituto di Scienza e Tecnologie dell'Informazione
"A. Faedo" (ISTI - CNR)
Area della Ricerca CNR di Pisa, 56100 Pisa, Italy
antonio.bucchiarone@isti.cnr.it

Stefania Gnesi

Istituto di Scienza e Tecnologie dell'Informazione
"A. Faedo" (ISTI - CNR)
Area della Ricerca CNR di Pisa, 56100 Pisa, Italy
stefania.gnesi@isti.cnr.it

Abstract

Composition of services has received much interest to support business-to-business (B2B) or enterprise applications integration. On one side, the business world has developed a number of XML-based standards to formalize the specification of Web services, their composition and execution. On the other side, the Semantic Web community focuses on reasoning about web resources by explicitly declaring their preconditions and effects with terms precisely defined in ontologies. So far, both approaches have been developed rather independently from each other. In this paper the major languages, namely BPEL4WS, BPML, WSCI, WS-CDL and DAML-S, are compared with reference to the requirements identified and finally the trend of Services composition is discussed.

Key words: Service-Oriented computing, Business-to-Business, Web Services, Web Services Composition, Semantic Web.

[1] IMT Graduate School, Via San Micheletto, 3 - 55100 Lucca , Italy

1 Introduction

Service Oriented Computing (SOC) is an emerging paradigm for distributed computing and e-business processing that finds its origins in object-oriented and component computing. Services are computational entities that are autonomous and heterogeneous (e.g. running on different platforms or owned by different organizations). Services are described using appropriate service description languages, published and discovered according to predefined protocols, and combined using an engine that coordinate the interactions among collaborating services. Web services technology is a widespread accepted instantiation of SOC which should facilitate integration of newly built and legacy applications both within and across organizational boundaries avoiding difficulties due to different platform, heterogeneous programming languages, etc.. Exploiting this kind of ubiquitous network fabric would result in an increasing of productivity and in a reduction of costs in *B2B* processes [10]. The business drive to increase the enterprise's agility in responding to customer and market needs has accelerated the integrations of various applications both within and across enterprise boundaries. Information flow and business processes must be streamlined and automated to increase overall business efficiency. Network connectivity with customers, suppliers and partners for quick and automated processing provides a vital and competitive edge for any enterprise. The idea behind this approach is to allow independently developed applications to be exposed as services and interconnected exploiting the already set up Web infrastructure with relative standards (HTTP [6], XML [12], SOAP [3] and WSDL [14]). *Web Services*, which are based on XML-based open standards, promise the interoperability of various applications running on heterogeneous platforms. They enable dynamic connections and automation of business processes within and across enterprises for enterprise application integration and business-to-business integration. Building on the ubiquitous and lightweight standards that are supported by mayor software vendors, Web services enable application integration via the publishing of application's functionality as services, as well as location and invocation of services over the Internet. Although Web services are sufficient for some simple interaction needs, they are not sufficient for integration of business processes that involve multiple enterprises. Business process integration in real business scenarios involve long-running interactions, transactions management, stateful invocations and are often driven by a workflow engine that execute a specified business process model to automate the information flow and the business operations. This raises the needs for *Web services composition* that provides the mechanism to fulfill the complexity of business processes execution. Different organizations are presently working on composition proposal. The most important in the past have been IBM's WSFL [8] and Microsoft's XLANG [7]. These two have then converged in Web Services Business Process Execution Language (BPEL4WS [4]) which is presently a working draft by OASIS. Another recent proposal in phase of standardization by the World Wide Web Consortium (W3C) is WS-CDL [21]. Both allow the definition of workflow-based composition of services with some similarities and some differ-

ences. These languages are designed to provide interoperability between various applications. The platform and language independent interfaces of the web services allow the integration of heterogeneous systems but, these web services standards do not deal with the dynamic composition of existing services. A more challenging problem is to compose services dynamically. In particular, when a functionality that cannot be realized by the existing services is required, the existing services can be combined together to fulfill the request. Currently the Web service technologies fall on the restricted capability to support static service composition. Their limit comes out from the total absence of semantic representation of the services available on the Internet. In response to these limitations, a number of solutions have been proposed by the Semantic Web community (for example DAML-S [17]).

The **Semantic Web** [13] is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. This is realized by marking up Web content, its properties, and its relations, in a reasonably expressive markup language with a well-defined semantics.

In this paper we present a survey of several Web Service Composition Languages. The objective of this paper is twofold. The first goal is to focus on the ability of each languages to: i) model the business collaboration, ii) model the execution control of processes, iii) represent the roles of participants, iv) manage transactions and compensations over services invocation, v) take into account exceptions handling, vi) have a semantic support, vii) support of Business agreement and viii) have a Software vendor support. The second is to take this analysis as the starting point in order to define the requirements of a new automatic framework for the design, analysis and development of composition in Service Oriented Computing. The structure of the paper is introduced in the next session.

1.1 Outline of the paper

The paper is structured as follows: after the above introduction, section 2 presents on overview of the various models for the Service-Oriented Composition. Section 3 introduce the current service composition languages from the different point of view of Web and Semantic Web services . It lists an overview of BPEL4WS, BPML, WSCI, WS-CDL and DAML-S languages while section 4 is devoted to a comparison on them respect to some requirements. Finally section 5 reports some conclusive considerations.

2 Service-Oriented Composition: Current Models

Composition of Services has received much interest to support business-to-business or enterprise application integration. The business world has developed a number of XML-based standards to formalize the specification of Web Services, their composition and execution. The Semantic Web community focuses instead on reasoning about web resources by explicitly declaring their preconditions and effects by

means of terms precisely defined in ontologies. The objective of this section is to introduce the existing composition models in the two different fields. We start with the definitions of Web Services and Semantic Web Services and then we will conclude with the introduction of various service composition models.

2.1 Web Services and Semantic Web Services

Web Services (WSs) are defined as self-contained, modular units of application logic which provide business functionality to other applications via an Internet connection. Web Services support the interaction of business partners and their processes by providing a stateless model of "atomic" synchronous or asynchronous message exchanges. Moreover WSs are characterized by two specifications: WSDL [14] and SOAP [3]. The former defines the interface that a Web service exhibits in order to be invoked by other services. The Web Services Definition Language (WSDL) is an XML-based language, which specifies a Web service by defining messages that provide an abstract definition of the data being transmitted and operations that a Web service provides to transmit the messages. Four types of communication are defined involving a service's operation (endpoint): the endpoint receives a message (one-way), sends a message (notification), the endpoint receives a message and sends a correlated message (request-response), and it sends a message and receives a correlated message (solicit-response). Operations are grouped into port types, which describe abstract end points of a Web Service such as a logical address under which an operation can be invoked. In Figure 1 we show a sample WSDL fragment of One-Way and Request-Response actions.

```
<wsdl : operation name="nmtoken">*
   <wsdl : input name="nmtoken"?
          message = "qname"/>?
   </wsdl : operation>
```

```
<wsdl : operation name="nmtoken">*
   <wsdl : input name="nmtoken"?
          message = "qname"/>?
   <wsdl : output name = "nmtoken"?
          message = "qname"/>?
</wsdl : operation>
```

Fig. 1. WSDL Operations: One-Way and Request-Response

WSDL provides a function-centric description of Web services covering inputs, outputs and exception handling.
The Semantic Web [13] provides a process level description of the service which, in addition to functional information, models the preconditions and postconditions of the process so that the evolution of the domain can be logically inferred. It relies on ontologies to formalize domain concepts which are shared among services. The Semantic Web efforts [13, 15], especially with respect to the recent trend towards **Semantic Web Services** [12], aim at fully automating all the stages of the Web services lifecycle. The Semantic Web considers the World Wide Web as a globally linked database where web pages are marked with semantic annotations. Semantic annotations are assertions about web resources and their properties expressed in the Resource Description Format (RDF) [18]. Along with RDF, one can use RDF Schema (RDFS) to express classes, properties, ranges and documentation

for resources and DAML-S [17] ontology to represent further relationships and/or properties like equivalences, lists, and data types.
With the Semantic Web infrastructure in place, practical and powerful applications can be written that use annotations and suitable inference engines to automatically discover, execute, compose, and interoperate Web services.

Given the different information that is available to specify a Web service in both the approaches we can subdivided the web services composition models in two classes: *Static* and *Dynamic* Services Composition (see figure 2).

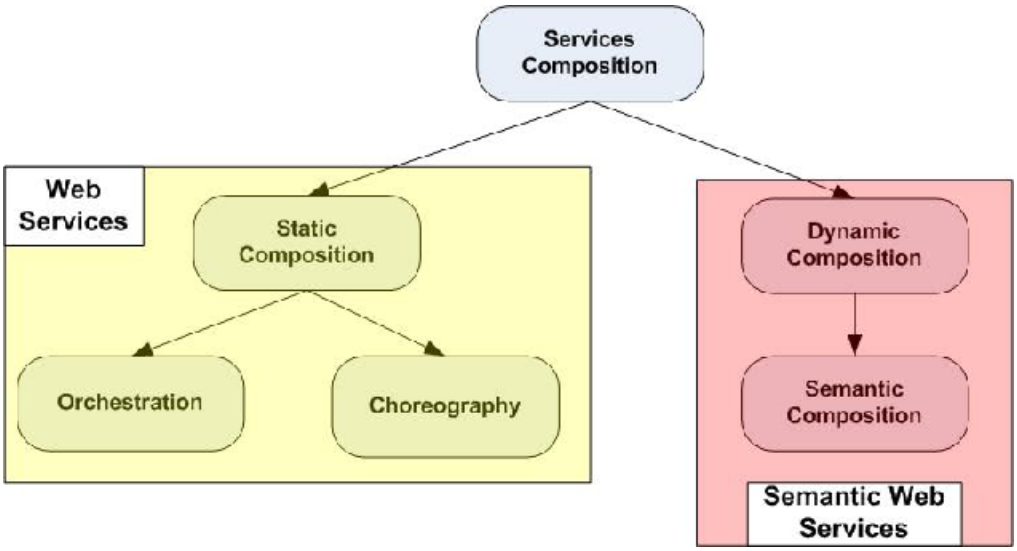


Fig. 2. Services Composition Models

2.2 Static Services Composition

A relevant feature for Web services is the mechanism for their reuse when complex tasks are carried out. It is often the case, to define new processes out of finer-grained subtasks that are likely available as Web services. To this aim, extensions of the Web service technology are considered which support the definition of complex services out of simpler ones. Composition rules deal with how different services are composed into a coherent global service. In particular, they specify the order in which services are invoked, and the conditions under which a certain service may or may not be invoked. Two possible approaches are currently investigated for the static service composition. The first approach, referred to as Web services ***orchestration***, combines available services adding a central coordinator (the orchestrator) which is responsible for invoking and combining the single sub-activities. The second approach, referred to as Web services ***choreography***, does not assume the exploitation of a central coordinator but it defines complex tasks via the definition of the conversation that should be undertaken by each participant. Following this approach, the overall activity is achieved as the composition of peer-to-peer interactions among the collaborating services. Several proposal already exist for orchestration languages (see e.g. BPML [19], BPEL4WS [4] etc..). On the con-

trary, choreography languages are still at a preliminary stage of definition. A first proposal, named WS-CDL [21], has been issued from W3C in December 2004.

2.3 Dynamic Services Composition

Web Services are designed to provide interoperability between different applications. The platform and language independent interfaces of the web services allow the easy integration of heterogeneous systems. Web languages such as Universal Description, Discovery, and Integration (UDDI) [5], Web Services Description Language (WSDL) [14] and Simple Object Access Protocol (SOAP) [3] define standards for service discovery, description and messaging protocols. However, these web service standards do not deal with the dynamic composition of existing services. The new industry initiatives to address this issue such as Business Process Execution Language for Web Services (BPEL4WS) [4] focus on representing composition where flow of the informations and the binding between services are known a priori. A more challenging problem it to compose services dynamically. In particular, when a functionality that cannot be realized by the existing services is required, the existing services can combined together to fulfill the request. The dynamic composition of services requires the location of services based on their capabilities and the recognition of those services that can be matched together to create a composition as described in [11]. The full automation of this process is still the object of ongoing research activity, but accomplishing this goal with a human controller as the decision mechanism can be achieved. The main problem for this goal is the gap between the concepts people use and the data computers interpret. This barrier can be overcome using semantic web technologies.

3 Overview of current service composition languages: from Web to Semantic Web

Web services composition that is guided by standards is an important element of the service-oriented computing. It enables broader interoperable business processes and provides reusability benefits. Various standards have been developed by major software vendors like IBM, Microsoft and Sun Microsystems. The process of refinement and convergence of standards are still going on. An overview of major Web and Semantic Web services composition language, namely BPEL4WS, BPML, WSCI, WS-CDL and DAML-S, is presented below.

BPEL4WS [4] is a XML-based specification language for specifying business processes that are exclusively based on Web services. It defines how multiple services can interact, giving the coordination rules of imported or exported functionality via Web services interface, necessary to achieve the business goals. It supports the definition of both executable business processes and abstract business processes by providing mechanisms to specify common core concepts of both types of processes with essential extensions for each process. An executable business process is defined as the model of the internal, actual behavior of a participant in a business

interaction, while abstract business process is defined as mutually visible message behavior of each party involved in business collaboration. BPEL4WS employs the concepts of partner links to directly model peer-to-peer collaborations between business processes and partners services. An instance of a business process can only be created when messages or events are received from partner links. Partner links define the static and abstract relations with other partners based on Web services *portType* used in interactions. BPEL4WS uses the notation of endpoint reference to dynamically select a provider for a particular type of service and to invoke the operation. The notation of Property is used to represent data elements within a message exchanged. A correlation set declares the correlation tokens that are used by BPEL4WS compliant infrastructures to build instance routing. BPEL4WS binds Web services into cohesive units encapsulated in activities.

BPML [19] is another standard proposed by Business Process Management Initiative (BPML.org). BPML was originally developed to enable the standard-based management of e-business processes used with Business Process Management System (BPMS) technology. It can be applied to a variety of scenarios, including enterprise application integration and Web services composition. BPML is a specification language dedicated to executable business processes. It is not surprising to see that BPML includes comprehensive support on control flow, data flow and event flow with structures of sending, receiving and invoking services, and control structures of conditional choice, sequential, iteration and parallel execution with synchronization. The definition of a Process is the template that instantiate process instances. It is the building block for modeling the execution of business processes. The concept of Signal is used to synchronize parallel activities with explicitly raising and waiting for the signal from activities. Exceptions are handled through the exception process and fault handler. Compensation is also supported to revert the effect of activities in processes. Transactions of business operation are supported with three means of declaring the activities as Atomic activity, Persistent process and Open nested transaction. The notation of Schedule is used to generate timing events that can trigger the instantiating of processes. BPML builds on top of WSDL for service description.

WSCI [20] was the first XML-based language that aims to provide a standard for specifying the overall collaboration between Web services providers and services users by describing messages exchanges that happen among the involving parties. WSCI only specified abstract business process that is observable between modeled Web services. WSCI was built on top of WSDL, which defines the Web services operation involved in WSCI activities. Actions are used to define basic requests and response messages. External services are invoked through Call. WSCI also supports the transactions and exceptions handling. WSCI addresses Web services-based choreography in two primary levels. At the first level, WSCI builds up on the WSDL portType capabilities to describe the flow of messages exchanged by a Web service in the context of a process. This, however, is still a

one-sided description of the message exchange, purely from the view point of the Web services own interface (portType). The construct introduced by WSCI permits the description of the externally "observable behavior" of a Web service, facilitating the expression of sequential and logical dependencies of exchanges at different operations in WSDL portType.

WS-CDL [21] is an XML specification targeted for composing interoperable, long running, peer-to-peer collaborations between Web services participants. It describes the global view of the observable behavior of messages exchange of all Web services participants that are involved in the business collaboration. WS-CDL is purely for abstract business processes specification, independent from the platforms and programming languages that are used to implement the Web services participation. WS-CDL models the peer-to-peer collaboration between participants with different roles using Choreography. Choreography makes use of Interaction and Activities notation to define the Relationships, which represents message exchanges between two Web services participants described in WSDL. Choreography supports exceptions and compensations with *Exception Block* and *Finalizer Block*. Messages that are exchanged between participants are modeled with Variables and Tokens, whose type can be specified in XML schema or WSDL. Channel is used to specify how and where messages exchange can take place. Activity performs the actual work of interaction through Interaction, which results in an exchange of messages. Perform invokes another Choreography, and Assign assigns the values to variables. Control structures that model the combined activities are simple, including Sequence, Parallel and Choice. Synchronization among activities can be achieved via WorkUnit, which defines the guard condition that must be fulfilled in order to continue Activity.

The **DAML-S**-services language (DAML-S) [17] is a set of language features arranged in ontologies to establish a framework within which the web services may be described in the semantic web context. DAML-S partitions a semantic description of a web service into three components: the service profile, process model and grounding. The *service profile* describes what the service does by specifying the input and output types, preconditions and effects. The *Process Model* describes how the service works; each service is either an AtomicProcess that is executed directly or a CompositeProcess that is a combination of other subprocesses. The *Grounding* contains the details of how an agent can access a service by specifying a communications protocol, parameters to be used in the protocol and the serialization techniques to be employed for the communication. The similarities between DAML-S and other technologies may be expressed as follows: The profile description has a similar functionality of the yellow pages in UDDI, the process model is similar to the business process model in BPEL4WS and grounding is just a mapping from DAML-S to WSDL. The main contribution of DAML-S is the ability to express the entities using the concepts defined in Semantic Web ontologies which provide expressive constructs that are suitable for the automatic discovery and composi-

tion of services. DAML-S service descriptions are made to link to other ontologies that describe particular service types and their features. The profile description of DAML-S services has a hierarchy as well. A profile hierarchy ontology describes this relationship and this information can be used for filtering the services that can be composed together.

4 A Comparison

We compare BPEL4WS, BPML, WSCI, WS-CDL and DAML-S according to the following requirements:

- *Modeling the collaboration*: The ability to perform long-lived, peer-to-peer collaboration between participating services. Collaboration must be modeled in terms of interactions of messaging exchanges.
- *Modeling the execution control*: The ability to assemble and incorporate individual Web services into the course of business process execution is vital for any Web services composition language.
- *Representation of Role*: Parties involved in business processes play different roles in different process stages. Representation of roles is necessary fo reflection of responsibility and behavior that parties are assumed in various scenarios.
- *Transactions and Compensations*: Business processes usually are long-running processes that may take hours or weeks to complete, and therefore the ability to manage transactions and compensations over services invocation is critical for Web services composition. Compensations are needed to rollback the effects of completed transactions when there is a failure in the enclosed transaction scope.
- *Exceptions handling*: The composition of Web services makes use of external Web services that are purely under the control of the Web services owner. It must take into account exceptions handling in the process of invocation when external Web services do not respond.
- *Semantic support*: Web services composition languages should enable the representation of semantics of composed services to facilitate the automated composition of Web services. The semantics descriptions that enable dynamic service discovery and invocation are imperative.
- *Business agreement support*: It is important in a business-to-business scenario to define agreement between involved parties. Business agreement defines the contract between two or more parties on Quality of Services (QoS). It is necessary to represent required QoS in composed Web services.
- *Software vendor support*: if the language has a Software support.

The comparison is listed in the table in Figure 3

In the comparison table the "Indirect" value means that the requirements are not directly supported by the language. For example in BPEL4WS the transactions are realized through faults handler and compensations. In summary all these languages provide the mechanisms to model the collaborations, only BPML needs of an in-

	BPEL4WS	BPML	WS-CDL	WSCI	DAML-S
Modeling the collaboration	Strong Support	Indirect support	Strong support	Strong support	Strong support
Modeling the execution control	Strong Support	Strong Support	No support	No support	Strong support
Representation of the Role	Weak support	No support	Strong support	Strong Support	No support
Transaction and Compensation	Indirect support	Strong support	Indirect support	Strong support	Indirect support
Exception handling	Strong support	Strong support	Support	Strong support	Strong support
Semantic support	No support	No support	No support	No support	Strong support
Business agreement support	No support	No support	No support	No support	No support
Software vendor support	Many	Few	No	Few	Few

Fig. 3. The comparison of BPEL4WS, BPML, WSCI, WS-CDL and DAML-S

direct support. BPML facilitates the modeling of execution of processes, while BPEL4WS and DAML-S attempt to cover both aspects. Finally WS-CDL is more natural to model B2B collaborations. All these languages support the imperative part of service composition, namely exceptions handling and compensations and all possess the capability to compose more complex structures and activities. The support for semantics representation and description is present only in DAML-S. Business collaborations require the established and enforcement of business agreement on QoS, but all these languages do not address this. As for tools support BPEL4WS has gained the widest support from industry. Most major software vendors has pledged BPEL4WS support in their products. Instead there is no too; support to WS-CDL, and it is not clear the industry acceptance of WS-CDL.

5 Conclusions and Future Work

There are other initiatives that work on the comparative study of composition languages [26, 27, 28]. Analysis of composition languages based on workflow pattern has also been done [29]. These comparison however are conducted almost at the micro level focusing on specific language like structure and control patterns. In this paper, an overview of major Web and Semantic Web services composition languages was presented. Five languages, namely BPEL4WS, BPML, WSCI, WS-CDL and DAML-S, were chosen and compared against eight requirements that a service composition language should support to facilitate the composing of business processes based on Web and Semantic Web services. DAML-S, like other service composition languages, provides a means to create description of Web services that can be interpreted programmatically. The distinguishing characteristic of DAML-S is that, while current Web services specification standards focus on service syntax, its goal is to facilitate the description of the semantics of services, their interfaces, and their behavior. The problem of the composition of web ser-

vices is addressed by two orthogonal efforts. From the one side, most of the major industry players propose low level process modelling and execution languages, like BPEL4WS [4]. These languages allow programmers to implement complex web services as distributed processes and to compose them in a general way. However, the definition of new processes that interact with existing one must be done manually, and this is a hard, time consuming, and error prone task. From the other side, research within the Semantic Web Community proposes a top-down unambiguous description of services capabilities, e.g. in DAML-S [17], thus enabling the possibility to reason about web services, and to automate web services tasks, like discovery and composition. This paper is an initial results and much more details must be describe. In future work we wish to extend the comparison with more details (e.g., more factual information as the tool support, messaging models supported etc.) in order to understand which language (BPEL4WS, BPML, WSCI, WS-CDL and DAML-S) or model (Static or Dynamic) for the composition is better than an other. An other key point that we would to deepen is the the QoS characteristics that each language is able to describe in order to define a QoS Service Composition. All this with the main objective to understand which are the elements of a framework for the automatic QoS composition of services.

6 Acknowledgments

This work was partly supported by the EU project IST-3-016004-IP-09 SENSORIA *(Software Engineering for Service-Oriented Overlay Computers)*.

References

[1] M.Aiello, M.P.Papazoglou, J.Yang, M.Carman, M.Pistore, L.Serafini and P.Traverso. *A request language for Web-service based on planning and constraint satisfaction*. In proceedings of the International VLDB Workshop on Technologies for E-Services, pages 76-85, Hong Kong, China, 2002.

[2] E.Christensen, F.Curbera, G.Meredith and S.Weerawarana. *Web Services Description Language (WSDL) 1.1*, 2001.
http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[3] W3C. SOAP 1.2 Working draft, 2001.
http://www.w3.org/TR/2001/WD-soap12-part0-20011217.

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

[5] UDDI Consortium. *UDDI Executive White Paper*, Nov. 2001.
http://uddi.org/pubs/UDDI_Executive_White_Paper.pdf.

[6] W3C. *HTTP: HyperText Transfer Protocol Specification*.
www.w3.org/protocols.

[7] Microsoft Corporation. *Microsoft BizTalk Server 2002 Enterprise Edition*, 2002. http://www.microsoft.com.

[8] F.Leymann. *Web Services Flow Language (WSFL) Version 1.0.* Technical Report, International Business Machines Corporation (IBM), May 2001.

[9] D.Burdett and N.Kavantzas. *WS choreography model overview.* Technical report, W3C (2004).

[10] N.Kavantzas. *Aggregating web services: Choreography and ws-cdl.* http://www.lists.w3.org/.

[11] Z.M.Mao, E.A.Brewer and R.H.Katz. *Fault-tolerant, Scalable, Wide-Area Internet Service Composition*. U.C. Berkeley TR UCB//CSD-01-1129, Jan 2001.

[12] S.McIlraith, T.C.Son and H.Zeng. *Semantic Web Services*. IEEE Intelligent Systems, 16(2), Mar. 2002.

[13] T.Berners-Lee, J.Hendler and O.Lassila. *The Semantic Web*. Scientific American, 284(5):34-43, May 2001.

[14] M.Dean, D.Connolly, F.van Harmelen, J.Hendler, I.Horrocks, D.L.McGuinness, P.F.Patel-Schneider and L.AStein. *Web Ontology Language (OWL) Reference Version 1.0*. W3C Working Draft 12 November 2002. http://www.w3.org/TR/2002/WD-owl-ref-20021112/.

[15] World Wide Web Consortium (W3C). *Semantic Web*. http://www.w3.org/2001/sw/.

[16] I.Horrocks et al. *DAML+OIL*, 2001. http://www.daml.org/2001/03/daml+oil-index.html.

[17] DAML Services Coalition. *DAML-S: Web Service Description for the Semantic Web*. In The First International Semantic Web Conference (ISWC), June 2002.

[18] D.Brickley and R.Guha. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation submitted 22 February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[19] BPML.org. *Business process Modeling Language (BPML)*. http://www. bpmi.org, 2002.

[20] A.Arkin et al. *Web Service Choreography Interface 1.0*, 2002. http://www.w3.org/TR/wsci.

[21] N.Kavantzas. *Web Services Choreography Description Language Version 1.0*. W3C, April 2004.

[22] T.Andrews et al. *Business Process Execution Language for Web Services (BPEL4WS)*, Version 1.1. http://ifr.sap.com/bpel4ws/index.html.

[23] Hewlett-Packard Company. *HP OpenView*. http://www.openview.hp.com.

[24] International Business Machine Corporation (IBM). *MQ Series Workflow for Business Integration*, 1999.
http://www.ibm.com.

[25] W.Baush, C.Pautasso and G.Alonso. *Programming for dependability in a service-based grid*. In Proceedings of the International Symposium on Cluster Computing and the Grid (CCGrid), Tokyo, Japan, May 2003.

[26] W.M.P.van der Aalst. *Don't go with the flow: Web Services composition standard exposed*. IEEE Intelligent System. January/February 2003.

[27] C.Peltz. *Web services orchestration, a review of emerging technologies, tools and standards*. January 2003.
http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrcherstration.pdf.

[28] J.Mednling and M.Muller. *A Comparison of BPML and BPEL4WS*, 2003.
http://wi.wu-wien.ac.at/mendling/publication/03-BXML.pdf.

[29] P.Wohed, W.M.P.van der Aalst et al. *Pattern Based Analysis of BPEL4WS*. Technical Report FIT-TR-2002-04,QUT, April 2002.

Session 2: Testing

Paper 4
Towards Model-Based Testing of Web Services
Lars Frantzen, Jan Tretmans, René de Vries
(*Radboud University Nijmegen - The Netherlands*)

Paper 5
Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN
José García-Fanjul, Javier Tuya, Claudio de la Riva
(*University of Oviedo - Spain*)

Paper 6
Initial Investigations into Interoperability Testing of Web Services from their Specification Using the Unified Modelling Language
Colin Smythe
(*eLoki - United Kingdom*)

Towards Model-Based Testing of Web Services

Lars Frantzen [1,3] Jan Tretmans [4] René de Vries [2,5]

Institute for Computing and Information Sciences
Radboud University Nijmegen
The Netherlands

Abstract

Complex interactions between Web Services involve coordinated sequences of operations. Clients of the provided services must be aware of the underlying *coordination protocol* to smoothly participate in such a coordinated setup. In this paper we discuss on a running example how such protocols may also serve as the input for Model-Based Testing of Web Services. We propose to use Symbolic Transition Systems and the rich underlying testing theory to approach modelling and testing the coordination. We further indicate where theoretical and technical gaps exist and point to several research issues.

Key words: Web Service, Coordination Protocol, Model-Based Testing, Symbolic Transition System

[1] Lars Frantzen carried out this work during is staying at ISTI/CNR supported by the Marie Curie Network TAROT (MRTN-CT-2004-505121) and by the Netherlands Organisation for Scientific Research (NWO) under project STRESS.

[2] René de Vries carried out this work as part of the TANGRAM project under the responsibility of the Embedded Systems Institute. Tangram is partially supported by the Netherlands Ministry of Economic Affairs under grant TSIT2026.

[3] Email: lf@cs.ru.nl

[4] Email: tretmans@cs.ru.nl

[5] Email: rgv@cs.ru.nl

1 Introduction

A Web Service (WS) never walks alone: it publishes interfaces to potential users of the provided service. To be utilised efficiently these interfaces need to be described in a way accessible to machines, i.e. in a language with formal syntax and semantics. In this paper we want to show how such interface specifications can be exploited to serve as the input for *Model-Based Testing* (MBT).

With testing we mean the act of performing experiments with the WS, aiming at either finding faults, or gaining adequate confidence in the conformance of the WS to its interface specification. Model-based testing is a kind of black-box testing, where these experiments are automatically generated from the formally described interface specification, and subsequently also automatically executed. By founding our testing approach on well established formal testing theories, we have the advantage of automating both test generation and test execution, and moreover, it is precisely defined what exactly we mean with conformance.

One crucial issue when following this path of model-based testing is the unavoidable trade-off between increasing *testability* and *information hiding*. By just publishing very general information about the provided service, e.g. mere signatures of the interface operations, one can completely hide the possibly proprietary business logic which constitutes the operations' implementation. Furthermore, the implementation can change freely as long as it still continues sticking to the interface description. Such kind of interface specifications can be expressed in the Web Service Definition Language (WSDL) [6]. When it comes to testing, the only information we can exploit in such a setting is the WSDL-file; we have no knowledge about any internal structure (e.g. the source code), nor about any ordering among the different operations. When testing in this setting, we can only test each provided operation separately, applying classical black-box testing techniques, such as equivalence partitioning or boundary value analysis.

This separate testing of each provided operation is fine as long as the operations the WS offers are independent from each other, i.e. all operations can be called at any time, without influencing each others behaviour. For instance, think of a WS translating Italian and Spanish words into English, thus offering two independent query operations. Testing such a WS boils down to testing each operation separately.

In many non-trivial WSs the invocations of operations are not independent; there are restrictions on the dynamics of operations, i.e., the invocations have to obey some ordering. We will use the term *WS protocol* to denote the specification of the legal orderings of invocations of a WS. Continuing the example above, the dictionary WS could be restricted to registered users. Thus, before being able to (successfully) invoke the query operations, a user first has to log into the system via a dedicated operation.

A mere collection of operation-signatures given in the WSDL only describes the static aspect of invocations; it is not sufficient to specify the allowed sequences of invocations. Thus, it is crucial that the WS offers an additional source of in-

formation, which can be accessed to discover the WS protocol a user has to obey. To make clients aware of the protocol, its description should be available in WS registries.

In order to proceed towards our goal of automatic, model-based testing of web services, WS protocols must be formally modelled. One commonly used model for specifying WS protocols is a state machine. We will propose to use a special variant of a state machine, namely a *Symbolic Transition System* (STS) as introduced in [7]. An STS has states and labelled transitions between states modelling the actions, i.e. the inputs and outputs, of the system. Both states and actions can be parameterised with data variables, and predicates on these variables may guard the transitions.

The use of STSs for specifications allows to exploit the well established STS-based testing theory and algorithms of [7]. These include a precise definition of conformance of a WS implementation with respect to its specification using the *implementation relation* **ioco** [13], an algorithm for the generation of test cases from an STS specification, and notions of soundness and exhaustiveness of the generated test cases. Moreover, several testing tools nowadays implement this test generation algorithm it, e.g. TorX [2] and the TGV-based tools [11]. The tool TorX generates and executes test cases on-the-fly, which means that instead of firstly computing a set of test cases from the STS, and then applying them to the System Under Test (SUT), it generates a single test event in each step, and immediately executes it to the SUT. As a consequence the state space explosion when generating test cases, is avoided, see also [2].

The STS model as defined in [7] turns out to be rich enough to formally model a WS protocol between a single client and web service; this will be elaborated in Sect. 3. More useful web services are commonly built by coordinating many simpler web services. This implies that we must take into account a WS protocol comprising several WSs which interact with each other. In Sect. 4 we will analyse the shortcomings of the currently defined STS model and its corresponding testing theory for testing these multi-WS interactions, and we will point to some first approaches to deal with these cases.

The goal of the paper is to discuss how current model-based testing techniques might be applied to testing web services, and to indicate where theoretical and technical gaps exist. It tries to motivate a promising path and its cornerstones, hopefully leading to a complete and formal treatment of testing web services in the future.

Overview

This paper is structured as follows. Section 2 introduces the running example used throughout the paper. In Sect. 3 we discuss the STS-model, and exemplify how to use it for modelling and testing the conversations between a single client and a WS. Section 4 shows further approaches to deal with a setting comprising several WSs. Section 5 concludes the paper.

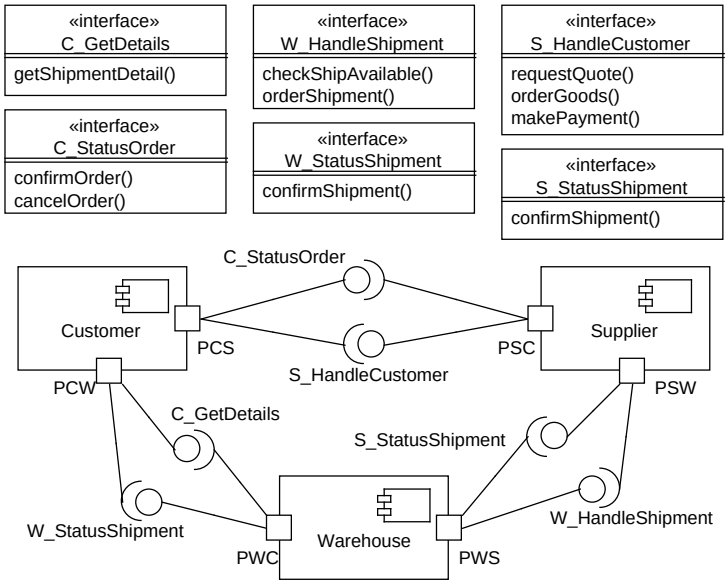


Fig. 1. The procurement protocol setup

2 Case: Procurement Scenario

In this section we briefly review the basics of WSs which are relevant for our MBT approach. For testing it is required to identify the Points of Control and Observation (PCOs), i.e. points where interaction occurs. Secondly, we need to identify the direction of interaction, i.e. whether there is an input (stimulus) or an output (observation). To analyse this, we model a WS in the realm of UML components, its ports and interface connectors. We review WSs by considering a running example of the Procurement Scenario adopted from [1].

Consider a customer who needs to order goods from a supplier. The customer orders at the supplier, which processes the order. Then the supplier requests a warehouse to deliver the goods to the customer. Now assume that we automate this scenario in a Business to Business setting, where every party has its own WS to interact with its peers. This is visualised in Figure 1 as an UML 2.0 Component Diagram.

We find three WSs `Customer`, `Supplier` and `Warehouse`. The points where we can connect to and interact with a WS are called *ports* (in Figure 1: `PCS`, `PCW`, `PWC`, `PWS`, `PSC` and `PSW`).

At a port we can execute *operations* by means of interaction. The direction of communication of these operations can be differently. We can distinguish among *one-way*, *notification*, *request-response*, and *solicit-response*. One-way can be considered as being able to accept a single input message from a peer WS. Notification can be seen as doing a single output message by the WS. Request-response corresponds to an API like invocation, i.e. the operation consists out of first accepting an input and then returning a result to the caller. The solicit-response operation can be considered as the WS taking initiative to make an invocation to another

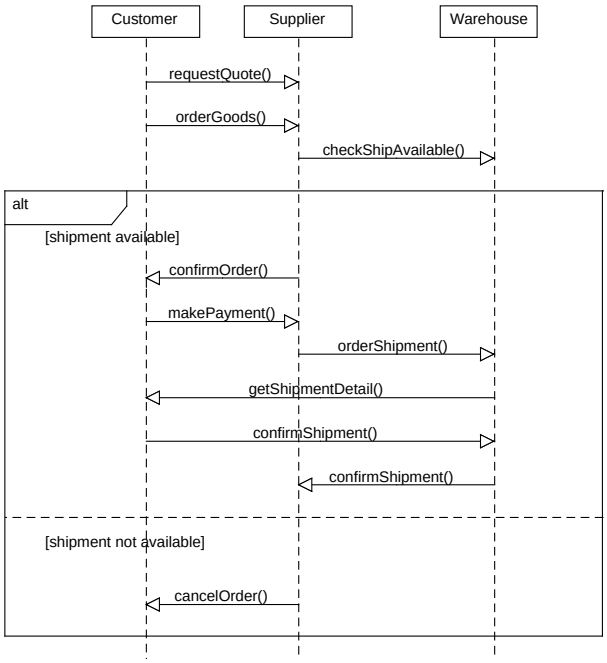


Fig. 2. The procurement protocol

WS, i.e. make a response (output) and waiting for an answer (input). Synchronous interactions are modelled via *one-way* and *notification* operations; asynchronous interactions correspond to *request-response* and *solicit-response* operations. In this paper we group operations at a particular port into *interfaces*.

Every port consists of several interfaces, e.g. `PCW` consists out of the interfaces `C_GetDetails` and `W_StatusShipment`. For now we consider an interface that takes initiative to communicate an *output interface* (i.e. solicit-response and notification operations), while an *input interface* is accepting messages (i.e. request-response and one-way operations). For the given Procurement Scenario we assume all operations to be synchronous, i.e. we only deal with solicit-response and request-response operations. For the `Customer` WS, `C_GetDetails` and `W_StatusShipment` are respectively an input and output interface. Typically the signature of the operations are described by WSDL. In our example the operation-signatures at the `W_HandleShipment` interface are `checkShipAvailable()` and `orderShipment()`.

The behaviour of a use case of the Procurement Scenario can be described by an UML 2.0 sequence diagram as is done in Figure 2. The `Customer` WS can connect to the `Supplier` WS to inform about the price, availability, delivery dates etc. of goods by means of the `requestQuote()` operation. Then it places the order by a `orderGoods()` operation. The `Supplier` WS checks at the `Warehouse` WS whether there is a shipment available by a `checkShipAvailable()` operation. Based on the result the `Supplier` WS responses the `Customer` WS a `cancelOrder()` operation (e.g. he ran out of stock) or confirms

the order by a `confirmOrder()` operation. If the supplier can deliver the following sequence occurs: the customer pays (`makePayment()`), the shipment is ordered to the warehouse (`orderShipment()`), the shipment details are provided to the customer (`getShipmentDetail()`), the customer confirms its order (`confirmShipment()`) and the warehouse confirms the shipment to the supplier (`confirmShipment()`).

3 Testing Conversations Between a Client and a WS

In this section we exemplify how to use STSs for modelling and testing the dynamics of conversations between a single client and a WS.

Coming to WS protocols, keeping the information hiding principle is not as straightforward as it is with mere operation signatures. The information revealed to the user depends on the chosen specification model, and on the part of the WS setup which is chosen as constituting the SUT. The more complex this part is, the more meaningful the cooperation of WSs can be tested. In the next subsections we start with very simple models. The first considers only a single input interface, where each operation invocation is modelled by an action. Then, in Sect. 3.3, we consider ports, but we then still model each invocation by a single action in the STS. Finally, starting in Sect. 3.5, we discuss testing where data is added to invocations, and each invocation is split into two actions: one to model the call or start of the invocation, and one to model the end or return of the invocation (also called *request* or *solicit*, and *response*, respectively.

3.1 Symbolic Transition Systems

Our specification models of choice are STSs. Such an STS is a state machine incorporating an explicit notion of data and data-dependent control flow (such as guarded transitions), founded on first order logic. The STS model clearly reflects the *Labelled Transition System* (LTS) model, which is done to smoothly transfer LTS-based test theory concepts to an STS-based test theory. The underlying first order structure gives formal means to define both the data part algebraically, and the control flow part logically. This makes STSs a very general and potent model for describing several aspects of reactive systems. We do not give here a formal definition of the syntax and semantics of STSs due to its extent, but motivate their usage in the following examples. For a formal definition we refer to [7].

3.2 Testing a Single Input Interface

Seeing a WS as being an isolated component, possibly neglecting the role it may play in a coordinated set up, a WS protocol may only refer to operations from the WSs input interface. Given a simple case where the conversation with the user does not happen via output interfaces, this might be a reasonable thing to do. Our example already shows that in practice for many non trivial examples a WS cannot be specified and tested from such an isolated point of view.

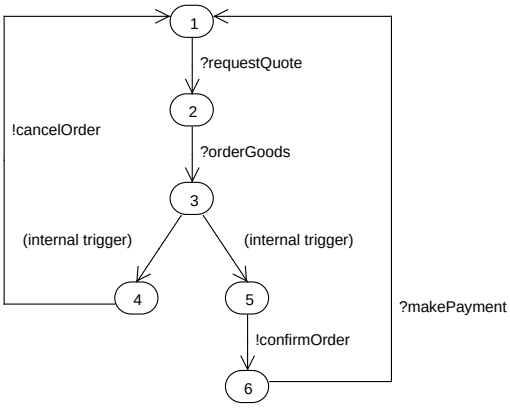


Fig. 3. A basic STS specification for the PSC port

Take for example the input interface S_HandleCustomer from the Supplier WS. Suppose one wishes to test the conversations between a Supplier and a Customer based on this interface. A WS protocol would specify that at first a customer has to invoke the requestQuote method, followed by an orderGoods method. From this point in time, one cannot express anymore if the client is supposed to initiate a makePayment call because this depends on the answer from the Supplier as being given via the output interface C_StatusOrder (which is an input interface for the customer). We have to observe if the supplier sends the message confirmOrder or cancelOrder to the customer to make a judgement here. Thus, a meaningful specification covering more than a prefix of a conversation cannot be given without considering at least the output interface C_StatusOrder.

Thence, a specification only comprising input interfaces does strictly uphold the information hiding principle, but does in general not constitute an entity one can utilise for MBT.

3.3 Specifying a Single Port

We will focus here on specifying the PSC port. This bi-directional port comprises the input interface S_HandleCustomer, and C_StatusOrder, the output interface. In this setting the tester plays the role of the Customer testing the Supplier. Thus, we can use the operations given in these two interfaces. Now we can express the protocol we were not able to give when restricting to just the input interface of the Supplier WS.

In Fig. 3 you find a first STS specification of the PSC port focussing only on specifying the legal ordering of the operations. In so doing we can neglect their return values since we are not interested in the data communicated. Thus, every operation in from the input interface is mapped to the input action ?in, and every operation out from the output interface is mapped to the output action !out. In this manner we can give the shown state machine and define the legal order-

ing of the operations. Note that there is one special label called `(internal trigger)`. This label corresponds to an internal action of the `Supplier` which is not observable in our current setting since it comprises only the `Customer` and the `Supplier`. It refers to the choice of either sending a `!cancelOrder` or a `!confirmOrder` to the `Customer`. Remembering the procurement protocol as being given in Fig. 2, this choice depends on the response from the `Warehouse` if the shipment is available or not. This part of the protocol is out of the scope in our current setting, hence we have to model this choice as a nondeterministic one. Such an `(internal trigger)` usually is referred to as an τ-action in the underlying theories.

3.4 Testing a Single Port

The given STS is a sheer specification of the order of the interface operations. The business logic revealed via such a specification of legal transitions is more than marginally. It is mandatory to at least inform potential user about the intended orders of operation calls to allow for a smooth communication. Since the data exchanged is completely omitted, the user has no access to knowledge like how it is determined by the `Supplier` if a shipment is cancelled or confirmed, etc.

The value of such a specification for testing a port is very limited. Since no information is given about the data exchanged via the operations, a test system has no access to any kind of indicators on how it should assemble or verify messages sent to, or received from the `Supplier`. Thus, such kind of specifications are more related to monitoring the conversations between WSs on the abstraction level of operation calls, auditing only the calls without looking further into the data exchanged. This can still be of high value to detect elementary communication faults in a coordinated setup.

3.5 Specifying a Single Port Comprising Data

In the basic STS example above we have not used any of the symbolic features of an STS. In fact, we could have modelled the same state machine as a simple LTS. To really test the specific details of a system we have to take into account the data exchanged. In the realm of distributed systems we have the complex situation of several ways and kinds which can be exploited to exchange data. For instance there can be synchronous and asynchronous message types. As we have seen an STS has in principle an asynchronous nature since every transition realises either an input action or an output action. This is fine since synchronous communication involves a tight integration and dependency of the interacting WSs, which is not acceptable in most industrial-strength settings. Still there might be cases where a synchronous setting is favoured, hence the applied modelling and testing techniques should support both kinds of interaction.

In the realm of STSs it is straightforward to model a synchronous operation as an input action representing the method call, followed by an output action representing the returned value (which might of type `void`, i.e. a mere return of

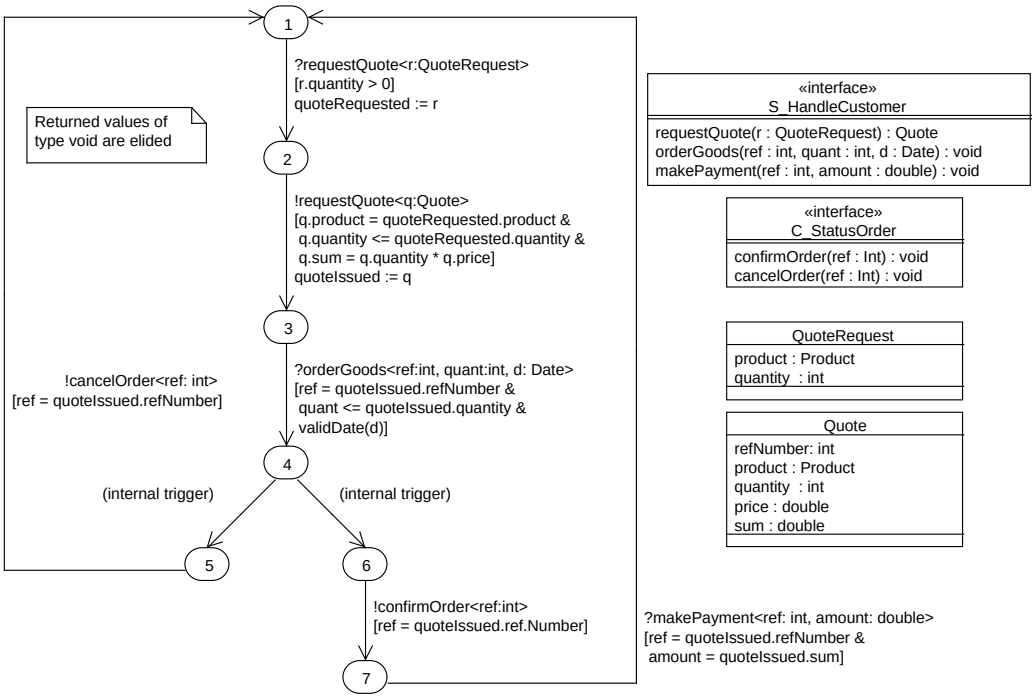


Fig. 4. A detailed STS specification for the `PSC` port

control). For instance take the `requestQuote` operation from the input interface. To model the data flow we need to know the complete signature of this operation, including types. The `requestQuote` operation gets an object of type `QuoteRequest` as input, and returns an object of type `Quote`. The complete signature is then `requestQuote(r:quoteRequest):Quote`. To model this operation in an STS we map it to the typed input action `?requestQuote<r:quoteRequest>`, and typed output action `!requestQuote<q:Quote>`. The inverse case shows up when the tested WS calls a method from the output interface. Here, the call is modelled as an output action, and the return is modelled as an input action. While in this two-party setting the splitting of a method call into two actions may appear as an redundant effort, we will see in the next section that the asynchronous nature of STSs is of utmost value when dealing with more then just two parties.

In Fig. 4 you find an STS specification of the `PSC` port specifying much more than just the legal ordering of the transitions. The data signatures of the corresponding interfaces are also given. Each STS transition consists of three parts: first the name of an input- or output action together with its parameters; next a guard talking about the parameters and the internal variables; and finally an update of the internal variables.

Such internal variables are a crucial concept for having a natural and powerful specification model. These concepts sometimes cause confusion when the strict duality between specification models and implementation models is overlooked. Of

course, a black box specification must not refer to the real implementation details like variables which really exists in the implementation. Specification variables like internal STS variables are used to abstractly model the state of the SUT and have not any kind of semantical relation to real variables from the SUT.

We exemplify the STS concept on the operation `requestQuote`. The invocation of the method corresponds to the transition from state 1 to state 2. First, the label mentions the input action `?requestQuote<r:QuoteRequest>`. Here we refer with `r` to the input value of type `QuoteRequest`. Next, the guard `[r.quantity>0]` constrains the attribute `quantity` of `r` to be a positive integer. The variable `r` is a special kind of variable, called interaction variable, which is local to an action, not global to the whole STS as internal variables are. Hence we have to save the value communicated via `r` in an internal variable `quoteRequested`. This is done via the assignment `quoteRequested:=r`.

The succeeding transition from state 2 to state 3 corresponds to the returned value of the `requestQuote` operation, which is an object of type `Quote`, referenced by the interaction variable `q`. The guard here ensures that returned quote deals with the same product as mentioned in the requested quote. It further constrains the offered quantity to be less or equal to the requested one, and ensures that the mentioned sum of the quote equals the quantity times price per item. This is done by relating the internal variable `quoteRequested` and the interaction variable `q`. Finally, the offered quote is saved in the internal variable `quoteIssued`.

We have to reconsider here the principle of information hiding. The presented STS clearly presents a more detailed view on the specified system. This is mainly due to the guards which reveal constraints we can assume when dealing with the system. These constraints are crucial for testing the system, but may not be crucial for mere coordination needs. It is up to the specific situation how detailed a specification is supposed to be, depending on which aim it serves.

3.6 Testing a Single Port Comprising Data

Given a specification of a port in the STS formalism, we can apply the test generation algorithm as given in [7]. The first action of a tester implementing that algorithm would be to request a quote from the `Supplier` for an arbitrary product with a positive quantity. Next, the tester will check the returned `Quote` object and verify that the guard of the concerned transition from state 2 to 3 is fulfilled. If this is not the case, the test stops with verdict `Fail`. For instance, this is the case when the quote names a quantity greater than the requested one. Given a valid return the tester sends next a suited `?orderGoods` message. This message returns the control without communicating data, these `void`-returns are elided in the STS.

Next, the tester does not know if the SUT is in state 5 or 6 due to a nondeterministic internal trigger. Hence, both receiving a `!cancelOrder` and a `!confirmOrder` message is conformant to the specification. In practice the testing continues in this manner until either a fault is discovered via verdict `Fail`, or the testing is stopped after a predefined halting criteria. Figure 5 shows a sim-

```
STS is in stable state - Choosen to give an input
Giving this testevent to the STS:
 02/05/06 18:47:33:650: ?requestQuote[QuoteRequest with quantity 551]
STS Manager: 1 instantiated location(s):
 Location: 2: [QuoteRequest with quantity 551, Quote with quantity 0]
Applying input to SUT.
Will apply output response from SUT to STS.
Giving this testevent to the STS:
 02/05/06 18:47:35:343: !requestQuote[Quote with quantity 505]
STS Manager: 1 instantiated location(s):
 Location: 3: [QuoteRequest with quantity 551, Quote with quantity 505]
STS is in stable state - Decided to wait for quiescence.
STS Manager: Found Quiescence.
Giving this testevent to the STS:
 02/05/06 18:47:35:765: (quiescence)
STS Manager: 1 instantiated location(s):
 Location: 3: [QuoteRequest with quantity 551, Quote with quantity 505]
STS is in stable state - Choosen to give an input
Giving this testevent to the STS:
 02/05/06 18:47:35:949: ?orderGoods[1, 438]
STS Manager: 1 instantiated location(s):
 Location: 4: [QuoteRequest with quantity 551, Quote with quantity 505]
Applying input to SUT.
Will apply output response from SUT to STS.
Giving this testevent to the STS:
 02/05/06 18:47:36:124: !orderGoods[]
STS Manager: 2 instantiated location(s):
 Location: 5: [QuoteRequest with quantity 551, Quote with quantity 505]
 Location: 6: [QuoteRequest with quantity 551, Quote with quantity 505]
STS Manager: Output found.
Giving this testevent to the STS:
 02/05/06 18:47:36:125: !cancelOrder[1]
...
```

Fig. 5. Testing the PSC port on-the-fly

plified extract of the output of a prototype tool implementing the exemplified test generation algorithm.

For the conformance testing in general it remains to be evaluated how such halting criteria should be defined. It will also depend on the given application domain and its inherent security demands which specific halting criteria is considered sufficient. There are several well known halting criteria for model-based testing, mainly concerning coverage of the specification ingredients (states, transitions, evaluation criteria for the guards, etc.). Also more specific testing scenarios (called test purposes) might be of high value.

4 Research Issues

In the previous section we have addressed the basics of testing WSs using MBT techniques. Mostly these techniques have been developed for testing reactive systems. In this section we discuss adaptations and extensions of the underlying theory to improve our approach for testing WSs in the full setup, consisting of multiple ports, multiple WSs, etc. Here we are confronted with the big picture as being given in Fig. 1. To keep the specifications simple we will focus again on basic STS models which only show the legal invocations of operations. Every argument presented here is equally valid for STSs comprising data.

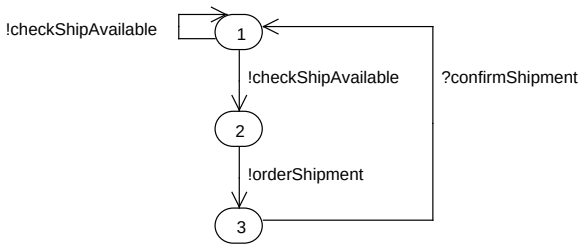


Fig. 6. A basic STS specification for the PSW port

4.1 Conformance Between Connected Ports

Following the running example we have focussed up to now on the `PSC` port of the `Supplier` WS. One obvious observation here is, that every port communicates with a partner port of another WS. For instance the `PSC` port talks with the `PCS` port of the `Customer`. Such partner ports are symmetric in the sense that both share the same set of interfaces, but what is an input interface for one is an output interface for the other. The `PSC` and `PCS` ports share the `C_StatusOrder` and `S_HandleCustomer` interfaces, in which `C_StatusOrder` is an input interface for the `Customer` and an output interface for the `Supplier`. The `S_HandleCustomer` is an output interface for the `Customer` and an input interface for the `Supplier`.

We may take advantage of this symmetry to define a notion of conformance between connected ports. Since connected ports have symmetric interfaces their underlying STS specifications also have symmetric actions. What is an input action for one is an output action for the other. Hence, swapping the set of input and output actions in such an STS yields a specification of its partner port. For instance one can swap inputs and outputs in the `PSC` specification of Fig. 4, i.e. visually all interrogation marks preceding an action become an exclamation mark, and vice versa. This resulting STS now constitutes a specification of the partner `PCS` port. So doing may allow us to formally define conformance of connected ports based on a formal conformance relation like **ioco**.

4.2 Integration of Service Specifications

A WS might consists out of several ports. Thus, if we want to test not only a single port, but a WS as a whole, we need to have specifications of all ports involved. In Fig. 6 the basic STS specification for the `PSW` port is given. This port together with the `PSC` port constitutes the `Supplier` WS. In principle both specifications could serve as the input for two independent test systems, each one testing a single port. But such a setting would ignore the logic combining the two ports. One of the assumptions of MBT is that a monolithic specification of the whole SUT, i.e. the WS protocol and signature, is available. However, when trying to integrate specifications corresponding to ports of a WS we are still leaking information about the dependencies between operations that occur at, for instance, the `PSC` and `PSW`

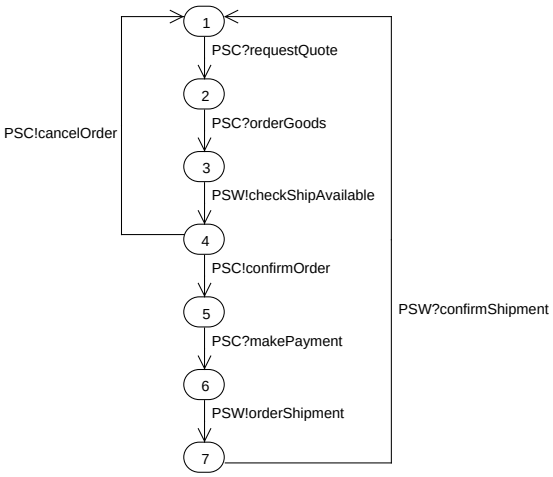


Fig. 7. A basic STS specification for the `Supplier` WS

ports. We propose here a research direction of defining service descriptions which allow for an aggregated model of the WS as a whole.

4.3 Specifications Embracing Several Ports

Having access to a service description covering the dependencies between provided and required services we can give a coordination protocol comprising a whole WS. Such an STS needs to make explicit at which port an action occurs. In Figure 7 we do so by preceding each action label by its corresponding port name. The shown STS combines both ports, `PSC` and `PSW`. Such a multi-port STS can then serve as the input for a tester covering all ports of a WS. For instance the upper specification empowers a tester to play both the role of the `Customer` and the role of the `Warehouse` when testing the `Supplier`.

The underlying theory on STSs is not yet ready to deal with such multi-port specifications. Though, there is a version of **ioco** called **mioco** [10] which can deal with several so called channels. One of our intended next steps is to define a similar extension for the STS theory to gain a theoretically sound approach for testing WSs in a coordinated setup.

4.4 Specifying Synchronous and Asynchronous Messages

In Fig. 8 it can be seen why the asynchronous nature of STS is of high value also in a setting of strictly synchronously communicating components. We can explicitly express and test for the order in which operation calls and returns appear. We exemplify the setup again from the view of the `Supplier` WS, receiving a `m1()` call from the `Customer` WS. This call is synchronous, hence during its duration the flow of control of the `Customer` is blocked. This focus of control is visualised in a sequence diagram by a thin rectangle showing the time during which an object is performing an action. During the execution of the `m1()` call the `Supplier`

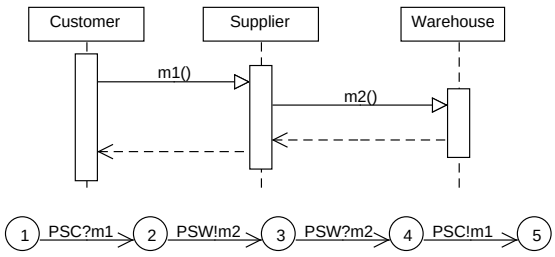


Fig. 8. Modelling synchronous calls with asynchronous actions

calls the `Warehouse` via the `m2()` operation. Having completed the execution of `m2()` the `Warehouse` sends a proper return message to the `Supplier` which in turn sends a return message to the `Customer`.

The given STS implements this order of operation calls and returns. Due to its asynchronous nature we can disperse input and output events in the shown manner. This is another important motivation to focus on asynchronous models like STSs, especially when dealing with a coordinated setup.

4.5 Input Enabledness

The underlying test theory in our approach (**ioco**) makes some strict assumptions about the implementation under test. One of these assumptions is that the implementation should be input enabled, i.e. under all conditions the system should be able to accept any input. A WS does not satisfy this requirement. Currently, we are revising the **ioco** theory and weaken this requirement. As a consequence we are now able to check whether an input is refused by an implementation. This means that when it is specified that a WS should accept an input message and it does not, we can detect it. Besides from theoretical soundness of applying the theory to WSs, we gain a stricter implementation relation resulting in a more rigorous assessment on the correct implementation of the WS. In other words, we can find more erroneous implementations.

4.6 Testing a Coordination

Finally coming to a coordinated setup comprising several WSs we encounter in principle the same defiances as when generalising from single ports to WSs: integration and more complex PCOs. Again we need a theoretically sound way to integrate service specifications of WSs, and to deal with the demands of a distributed testing scenario.

5 Conclusions and Related Work

When applying well-established theories and tools for MBT of reactive systems in the realm of WSs, one is confronted with the specific originalities of this domain. To analytically approach these new demands we model a setup of communicating

WSs as an UML Component Diagram, making the entities explicit which are relevant for establishing communication between WSs. In this setting a set of ports constitutes a WS. Such a port turns out to be the smallest meaningful entity we can identify for serving as a PCO. For specifying such a port the STS formalism is highly suited due to its asynchronous nature. We have also motivated the embedding of synchronous interactions. By so doing we can apply the rich testing theory as described in [7].

A specification of a WS comprising several ports cannot be given as a standard STS anymore since the actions have to be mapped to the ports where they occur. This is more than just a mere technicality since concepts fundamental to the underlying theory like input-enabledness have to be reconsidered. We made these issues explicit to give a clear picture of the problems to be tackled in future research.

Thus, having a complete and formal treatment of testing WSs is still a long way to go. We have proposed a direction which is founded on theories and tools which have proven to be of high value in the domain of reactive systems. As we see it now there is a promising chance that this fundament can be adopted to scale up into the realm of coordinated WSs.

There are other aspects when specifying and testing WSs which might be of high relevance in specific application domains. For instance expressing and testing timing constraints is an obvious candidate here. First promising results in extending the **ioco** theory in this direction have been published [5].

In [3] we have proposed to enrich the interface specification with an UML *Protocol State Machine* (PSM) [12]. We could motivate the transformation of such a PSM into an STS. The whole process was embedded in a framework for WS testing called *Audition* [4]. PSMs basically reflect the need to specify legal orderings of invocations for a specific interface or port. It turned out that mapping a PSM to a strict formal model like an STS is far from being straightforward. It is one branch of our current interests to find an adequate way of using UML models on top of STSs.

In [9] the authors propose to include graph transformation rules that will enable the automatic derivation of meaningful test cases that can be used to assess the behaviour of the WS when running in the "real world". To apply the approach they require that a WS specifically implements interfaces that increase the testability of the WS and that permit to bring the WS in a specific state from which it is possible to apply a specified sequence of tests.

The idea of providing information concerning the right order of the invocations can be found in a different way also in specifications such as BPEL4WS and the Web Service Choreography Interface (WSCI). The use of such information as main input for analysis activities has also been proposed in [8].

References

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

[2] A. Belinfante, J. Feenstra, R.G. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal test automation: A simple experiment. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, 12^{th} *Int. Workshop on Testing of Communicating Systems*, pages 179–196. Kluwer Academic Publishers, 1999.

[3] A. Bertolino, L. Frantzen, A. Polini, and J. Tretmans. Audition of web services for testing conformance to open specified protocols. In Ralf Reussner, Judith Stafford, and Clemens Szyperski, editors, *Architecting Systems with Trustworthy Components*, number 3938 in LNCS. Springer-Verlag, 2006.

[4] A. Bertolino and A. Polini. The audition framework for testing web services interoperability. In *Proceedings of the 31st EUROMICRO International Conference on Software Engineering and Advanced Applications*, pages 134–142, Porto, Portugal, August 30th - September 3rd 2005.

[5] L. Brandán Briones and E. Brinksma. Testing multi input-output real-time systems. In *ICFEM 2005 Seventh International Conference on Formal Engineering Methods.*, Manchester, UK, Nov 2005. Springer.

[6] E. Christensen et al. Web Service Definition Language (WSDL) ver. 1.1, March 2001.

[7] L. Frantzen, J. Tretmans, and T.A.C. Willemse. Test generation based on symbolic specifications. In J. Grabowski and B. Nielsen, editors, *FATES 2004*, number 3395 in LNCS, pages 1–15. Springer-Verlag, 2005.

[8] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL web services. In *Proc. of WWW2004*, May, 17-22 2004. New York, New York, USA.

[9] R. Heckel and L. Mariani. Automatic conformance testing of web services. In *Proc. FASE*, Edinburgh, Scotland, Apr., 2-10 2005.

[10] A. W. Heerink. *Ins and Outs in Refusal Testing*. PhD thesis, University of Twente.

[11] C. Jard and T. Jéron. TGV: theory, principles and algorithms. In *IDPT '02*, Pasadena, California, USA, June 2002. Society for Design and Process Science.

[12] Object Management Group. *UML 2.0 Superstructure Specification*, ptc/03-08-02 edition. Adopted Specification.

[13] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 17(3):103–120, 1996.

Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN

José García-Fanjul, Javier Tuya, Claudio de la Riva [1,2]

Computer Science Department
University of Oviedo
Gijón, Spain

Abstract

Generating test cases for compositions of web services is complex, due to their distributed nature and asynchronous behaviour. In this paper, a formal verification tool – the SPIN model checker – is used to generate test suite specifications for compositions specified in BPEL. A transition coverage criterion is employed to define a systematic procedure to select the test cases. The approach is applied to the "loan approval" sample composition.

Key words: web service compositions, model-based testing, structural testing, model checking.

[1] This work is supported by the Ministry of Science and Education (Spain) under the National Program for Research, Development and Innovation, projects IN2TEST (TIN2004-06689-C03-02) and REPRIS (TIN2005-24792-E)

[2] Emails: {jgfanjul, tuya, claudio}@uniovi.es

1 Introduction

From the inception of the Web, a growing number of companies have tried to use it as a new commercial channel. To start with, the most visible approach to leveraging this technology was the deployment of publicly available web-based shops, committed to collecting orders made by consumers. At an early stage, businesses also saw the opportunity to establish commercial electronic relations among each other. To do so, they needed to agree on a certain protocol to exchange information. This was one of the reasons for the W3C to develop the Extensible Markup Language (XML) [26]. Using this technology, businesses would agree to exchange data on a certain xml-based format or even follow public standards for data interchange such as ebXML [22]. The high acceptance of XML led to the development of software components that exclusively take this language as input, and also produce XML output: XML-based web services. These services are, furthermore, distributed in nature, asynchronous, low-coupled and platform-independent. Their composition enabled the implementation of interoperable business processes. To standardize the specification of these compositions, IBM and other companies proposed a language called BPEL [14] that later became an OASIS standard [23]. All these technological changes have been encouraged by an increasing investment in web services software worldwide, which doubled from 2003 to 2004, reaching $2.3 billion. That figure is expected to continue to grow and become $15 billion by 2009, according to IDC research studies [15].

However, this increasing interest in web services has also led to concerns regarding their trustworthiness. For example, Leavitt [16] highlights the differences between competing standards and Zhang [27] the lack of appropriate development and testing processes. In relation to the latter, the main characteristics of web services influencing testing activities are their asynchronous behaviour, distribution, availability and the lack of user interface.

Bearing in mind these features, the goal of this paper is to describe a model-based testing method to obtain test case specifications for compositions of web services. The BPEL language is introduced in Section 2 along with the composition that will be used as a sample throughout the paper: the "loan approval" composition. An overall description of the test generation method is then given in Section 3. A procedure is described in Section 4 to transform the composition specified in BPEL into a PROMELA model, which is the input language of the SPIN model checker [12]. After that, in Section 5, the approach employed to generate test cases specifications is explained and applied to the sample composition. To end the paper, Section 6 expounds related work and Section 7 details the conclusions and future work.

2 Specification of web services compositions with BPEL

BPEL specifications represent the behaviour of business processes based on compositions of web services. They are XML documents composed of two main sec-

```
<process name="loanapproval" [...]>
  <variables>
    <variable name="riskAssessment"
               messageType="asns:riskAssessmentMessage"/>
  [...]
  </variables>
  <partners>
    <partner name="customer" [...]/>
    <partner name="assessor" [...]/>
    <partner name="approver" [...]/>
  </partners>
  <flow>
    <links>
      <link name="receive-to-assess"/>
      <link name="assess-to-setMessage"/> [...]
    </links>
    <receive name="receive1" partner="customer" [...]>
    [...]
    </receive>
    <invoke name="invokeAssessor" partner="assessor"
          portType="asns:riskAssessmentPT"
          operation="check"
          inputVariable="request"
          outputVariable="riskAssessment">
      <target linkName="receive-to-assess"/>
      <source linkName="assess-to-setMessage"
              transitionCondition=
              "bpws:getVariableData('riskAssessment',
              'risk')='low'"/>
      <source linkName="assess-to-approval"
              transitionCondition="
              bpws:getVariableData('riskAssessment',
              'risk')!='low'"/>
    </invoke> [...]
  </flow>
</process>
```

Fig. 1. Extract from the "loan approval" BPEL specification

tions: declarations and the specification of the business process itself. In the declarations part, the `partners` are identified: each partner stands for a web service that participates in the business process. Other elements included in this first part are the `variables`, which enable the intermediate storage of values.

The specification of the business process consists of a set of activities that can be executed. These activities may be either basic or structured. Among the first, the business process can invoke web services or provide operations by means of the `invoke` and `receive` activities. It can also update the values of the variables using the assign construct. Structured activities prescribe the order in which a collection of activities take place. For example: a `sequence` activity establishes a sequential order and a `while` forces the repetition of the execution of a set of activities until a given statement becomes false.

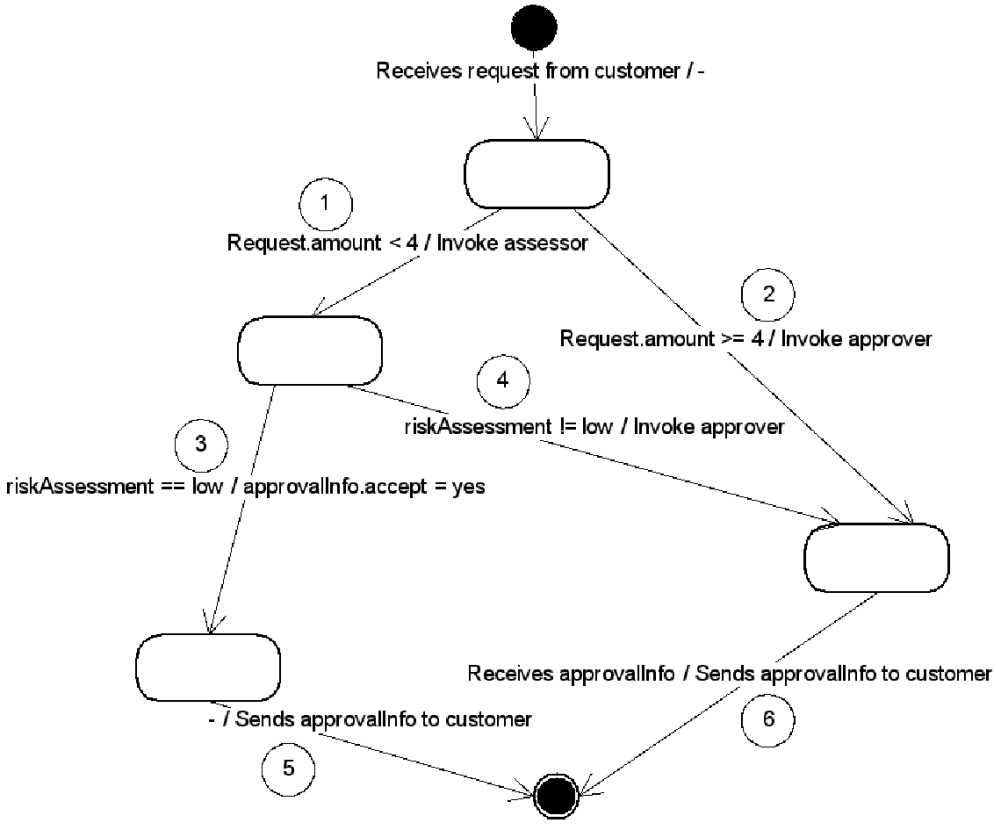


Fig. 2. Representation of the BPEL process described in the "loan approval" sample

A structured activity that is not so common in other languages is the `flow`. Activities grouped in such a construct are concurrent, and so a flow completes when all of its activities have completed. BPEL activities may also be assigned a construct that expresses synchronization dependencies between them: `link`. If activity A is one `source` of a link and activity B is one of its `targets`, that means B must be executed after A. Links may also be given a `transitionCondition` attribute that specifies a necessary condition for the link to become active.

A simplified version of the sample BPEL composition called “loan approval” is outlined in Figure 1. It was published within the specification of the standard [14] and is frequently applied to validate research on web services development. The goal of this business process is to conclude whether a certain request for a loan will be approved or not. To do so, it receives a request from a partner called “`customer`” and invokes two other partners. The “`assessor`” partner measures the risk associated with low amount requests. Another partner, called “`approver`”, approves requests that are either made for a large amount of money or which are evaluated by the assessor as not having a low risk. Figure 2 represents the behaviour of this business process.

3 Overview of the generation of test cases specifications

Model checking [4] is a formal verification technique that enables the automatic detection of whether certain properties hold in a model. It has lots of well documented applications, ranging from the verification of protocols [25] to fault detecting in software systems [11]. SPIN is one of the most commonly used model-checking tools. Using SPIN, properties can be specified by assertions in the model or shaped as Linear Temporal Logic (LTL) formulae. The tool searches all the possible states within the model and checks whether the properties hold. If not, it gives a trace of the steps illustrating the violation of the property, which is called a counterexample. Model checking is commonly used for systems verification, but it can be applied to generate test cases [24, 10]. In order to obtain a test case for a certain condition C, the model checker is fed with a model for the software and a LTL formula stating that C never holds. The output obtained from the tool is hence a counterexample in which the software fulfils C. That counterexample can be transformed into a test case, as it describes an execution of the software in which the desired test condition holds.

The above technique can be applied to generate test case specifications for BPEL compositions. Firstly, the Composition Under Test (CUT) is transformed into PROMELA. Then, in order to produce test cases, test requirements are identified using a transition coverage criterion. To do so, transitions are identified in the BPEL specification, whether explicit or implicit, and are mapped onto the model. In addition, each transition is expressed in terms of a LTL property expressing that "`the transition X is never executed`". The counterexample obtained from a SPIN run is thus a sample execution of the BPEL process in which at least the transition included in the LTL is exercised. To obtain a set of test cases that provides transition coverage, the tool is repeatedly executed with each previously identified transition. This method is depicted in Figure 3 and will be described in further detail in the following sections.

4 Transforming BPEL specifications into PROMELA

As explained in Section 2, BPEL specifications express the behaviour of a business process. This behaviour is modelled, in PROMELA, into a `process` construct. The specifications also include partners, which are the different web services. They are also transformed into PROMELA processes: one is included for each partner in the specification.

The business process and the partners communicate through portTypes, which are transformed into PROMELA message channels as in the following example:

```
chan loanassessor_riskPort_IN = [QLENGTH] of
                                {byte, byte, byte};
```

In the above example, a channel called `loanassessor_riskPort_IN` is declared with a maximum length of QLENGTH messages and supporting messages

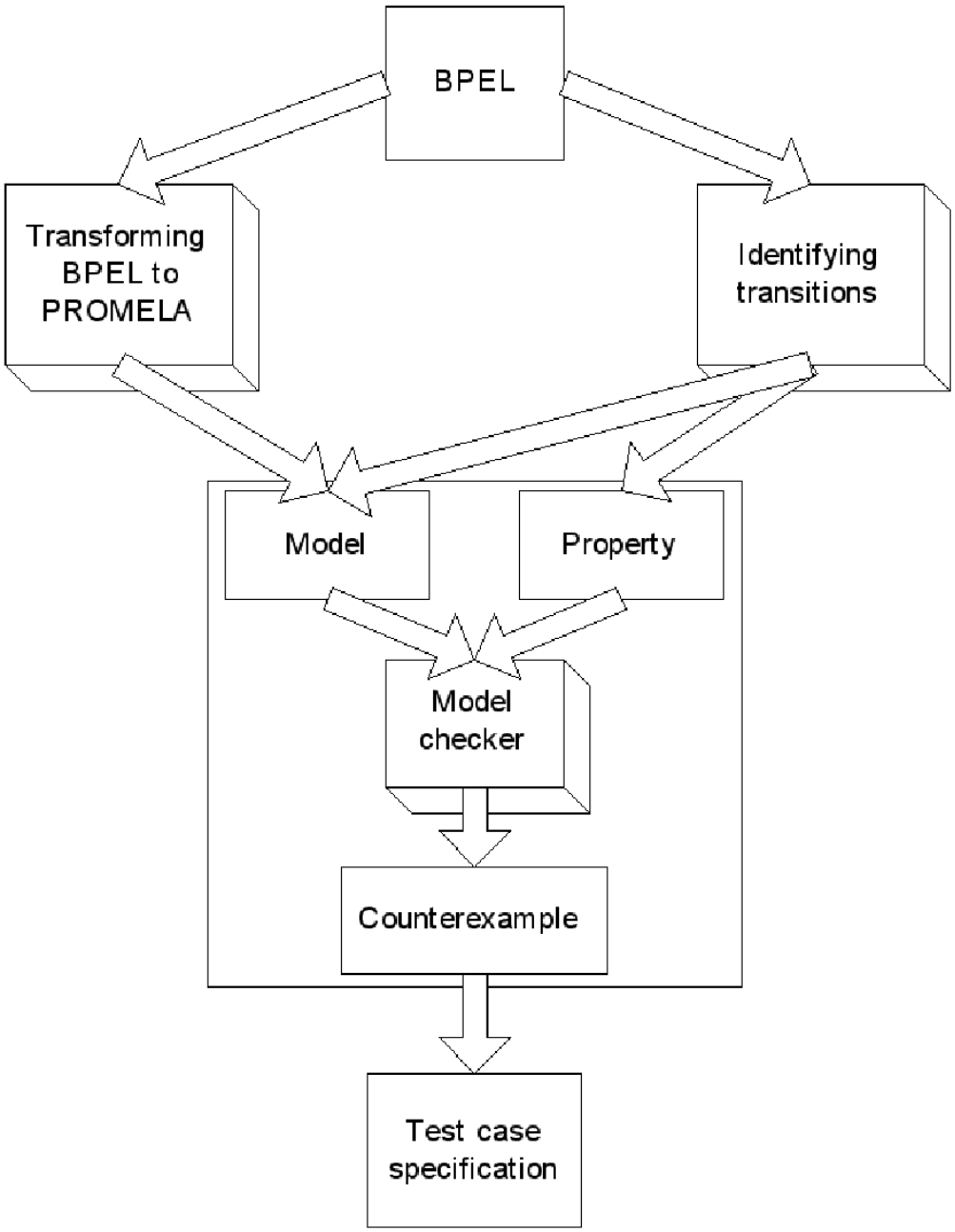


Fig. 3. An overview of the proposed method

consisting of three attributes of type `byte`. Each portType will have two channels: one for input messages and another for output.

Message types are declared as `typedefs` in PROMELA (a construct similar to "record" in the C programming language). BPEL data types are discretized from the original ones: every simple data type becomes byte except for boolean, which stays the same.

Each activity must be appropriately modelled to represent the behaviour of the business process. For example, `invoke` and `receive` activities use the `!` and `?`

channel operators that send and receive messages to and from the channels. Therefore, the statement `BPEL_loanApprovalPort_OUT! approvalInfo.accept` sends a message containing the variable `approvalInfo.accept` to channel `BPEL_loanApprovalPort_OUT`, which is the representation of the output portType of the business process in the "loan approval" sample.

In our approach, the behaviour of the partners is modelled using the BPEL specification as the only input. This is one of the major differences with other lines of work such as [8]. To do so, two sources of information about the partners' behaviour exist in BPEL. Firstly, the kinds of operations that the partners provide or request and the exchanged data are explicit in the specification. Secondly, in a more indirect way, BPEL specifications include how the business process handles the information received from or sent to the partners. This information enables us to build a simple PROMELA model (a mock) for each partner, which is incorporated into its PROMELA process. For example, the business process of the "loan approval" example receives `riskAssessment.risk` from partner `assessor`, and then examines whether it is `low` or not. Consequently, `assessor` is modelled to reply `low` and `other` in order to exercise both conditions. As a result, the behaviour incorporated into the PROMELA process for a partner follows these rules:

(i) If the BPEL specification has no reference to the data, it will be given an `undefined` value;

(ii) if the data is compared to a numerical constant, it will be given the value of the constant, a lower value and a higher third value;

(iii) if the data is discrete, it will be given each of the discrete constants in the BPEL specification and a value different from them, called `other`.

After the BPEL specification is modelled in PROMELA, transitions are identified in the specification and mapped within the model. Two kinds of transitions are distinguished: implicit and explicit. The first ones are obtained from activities that impose at least two possible execution paths. For example, an `invoke` activity may be defined to receive a reply from the invoked web service or not, so two transitions can be identified. Other activities with implicit transitions are `flow` or `while`. Explicit transitions are taken, in the composition, from `link` constructs, as they explicitly establish transitions between activities that have them as `source` or `target`.

5 Using SPIN to generate test case specifications

The test cases generated with SPIN will be systematically selected to satisfy a transition coverage criterion for the BPEL specification. Specifically, the criterion (taken from [20]) states that the resulting test suite must include test cases that cause every transition in the BPEL specification to be taken. To generate test cases specifications for the CUT that satisfy such a criterion, two different steps are required. First of all, LTL properties must be specified for SPIN to find counterexamples in which the PROMELA transitions are covered. Secondly, counterexamples must be

transformed into test cases specifications for the CUT.

All the identified transitions are uniquely differentiated by means of a number and, in PROMELA, a boolean variable called `tranX` (where X is the number of the transition) is defined for each of them. The variable will take a positive value in the case of the transition being exercised by the model checker. Those variables are employed to specify a LTL property and find a counterexample for one given transition. Namely, for transition X, the LTL is "`[] !tranX`", expressing that the variable associated with that transition is always false. After the model checker has been executed, a certain counterexample is given in which the transition is exercised.

In order to complete the test case specification, a difficulty arises when trying to interpret the expected output of the test case. The obtained counterexamples give only the sufficient indications so as to reach a state in which the property is false, but do not continue the execution of the model to its end. Therefore, to get the expected output for the test case, a boolean variable is included within the model, indicating that the business process ends. Thus, the specifications of the properties change to LTL formulae such as:

"`[](!tranX || !bpel_ends)`"

Lastly, to obtain the test case specification, it suffices to take the operations with channels. These operations give a complete description of the test case inputs and the output, as they represent the information exchanged between the business process and the partners.

To build a test suite that meets the above defined transition coverage criterion, the model checker is executed as many times as transitions are identified in the BPEL. To reduce the number of test cases, all the transitions covered with each counterexample are taken into account. In the described model, this is accomplished by inspecting the values of all the variables associated with transitions.

Applying the method to the "loan approval" composition, the numbered transitions in Figure 2 are identified and mapped into PROMELA. Using SPIN with the model obtained from the BPEL and completing the identified transitions, we need three executions to obtain transition coverage. On the first run, the model checker is fed with the LTL "`[](!tran1 || !bpel_ends)`" and so it produces the counterexample represented in Figure 4. In this figure, the partner that executes the step is shown on the left – customer, bpel or assessor. The counterexample covers transitions «1, 3 and 5» and is transformed into a test case specification with two inputs:

(i) the customer makes a request for an amount of 3 (less than four)

(ii) the risk assessment from the assessor is low

and one expected output: the reply to the customer is affirmative.

After this first result, SPIN is run twice again, finding counterexamples that cover transitions «2 and 6» and «1, 4 and 6», respectively. These counterexamples are transformed into test cases as in the example presented above. In this case

```
customer: request.amount = 3
customer: BPEL_loanApprovalPort_IN!request
bpel: request.amount<4
bpel: tran1 = true
bpel: loanassessor_riskAssessmentPort_IN!request
assessor: riskAssessment.risk = low
assessor: loanassessor_riskAssessmentPort_OUT! riskAssessment
bpel: tran3 = true
bpel: approvalInfo.accept = yes
bpel: tran5 = true
bpel: BPEL_loanApprovalPort_OUT!approvalInfo
bpel: bpel_ends = true
```

Fig. 4. Extract from a counterexample obtained by SPIN

study, the number of test cases obtained is the minimum required to give transition coverage for the specification.

As regards the performance of the tool, the execution of the verification ends in less than one second on a Pentium4 (3.0 GHz) system with 2 GB of RAM memory, using 32 internal states with a state-vector of 96 bytes to represent the model.

6 Related work

Research in verification and validation applied to compositions of web services may be basically classified in two categories: papers describing formal verification approaches and others that use testing techniques.

Related work describing verification approaches is quite common. The goal is to decide whether it may be said that certain properties hold in the composition under study. These approaches thus share with ours the use of verification tools and the need to build a model for the composition. Fu et al [8] define such a model and also use SPIN to formally verify compositions of web services specified in BPEL. They subsequently enhanced their research to define formal criteria for the feasibility of automated verification applied to compositions of web services [9, 3]. In the same line of work, Foster et al [6] use Finite State Processes (FSP) to model compositions of web services and describe the use of the LTSA tool [7] to formally verify BPEL specifications. They propose specifying the desired properties in terms of Message Sequence Charts, a technique included in the Unified Modelling Language (UML). Lerner [18] also uses the LTSA tool to analyze business processes specified in Little-JIL. Using a different model and verification paradigm, Narayanan and McIlraith [19] propose annotating web services with semantic descriptions (DAML-S) regarding their capabilities to subsequently encode these in a Petri Net. Another, not so closely related paper by Arias-Fisteus et al [1] describes an intermediate formalism (Common Formal Model or CFM) that can be used to specify business processes or to translate existing specifications into it. Subsequently, CFM models are automatically transformed into the input languages of existing verification tools.

There are not many research works on the definition of testing methods for web

services. Chun and Offutt [17] and Offutt and Xu [21] describe the application of mutation analysis and data perturbation in the testing of web services. Their processes are defined at the unit level, so the targets are the individual web services and not their composition. Bertolino and Polini [2] propose a framework for dynamic testing of web services interoperability. They introduce a test phase (an audition) before the services are published on a UDDI registry. In combination with verification techniques, Huang et al [13] describe a method whose goal is similar to ours: the testing of composite web services. They also apply a model checker, but the main differences lie in the input and the testing criteria: they explicitly specify the web services behaviour (using OWL-S) and define the desired properties by hand.

7 Conclusions and future work

In this paper, a model-based method is expounded for obtaining test cases specifications from BPEL compositions of web services to fulfil a transition coverage criterion. The method relies on a model checking tool (SPIN) to automatically obtain test cases specifications from a model of the BPEL process and partners. LTL properties are properly constructed for the resulting test cases specifications to cover transitions identified in the input. After repeated execution of the tool, a test suite is obtained that achieves transition coverage. In the case of the specification having unreachable transitions, the model checker automatically detects these by not providing a counterexample and performing a full verification. One of the advantages of the proposed method is its independence from the particular implementation, as the only required input is the BPEL specification.

An immediate line of future work on the described method is the application of different test criteria, such as those described by Offutt et al in [20] and its automation. Further research is likewise needed to fully determine the scalability of the method and, as there are no publicly available real-life specifications [5], synthetic ones may need to be constructed and tested. To automatically build test cases that are directly executable, the model could be enhanced so as to include information about the partners' particular implementation.

References

[1] Jesús Arias-Fisteus, Luis Sánchez Fernández, and Carlos Delgado Kloos. Formal verification of bpel4ws business collaborations. In *Proceedings of 5th International Conference on E-Commerce and Web Technologies*, volume LNCS 3182, pages 76–85, Zaragoza, Spain, August 31-September 3 2004. Springer Verlag.

[2] Antonia Bertolino and Andrea Polini. The audition framework for testing web services interoperability. In *Proceedings of 31st EUROMICRO*, pages 134–142, Porto, Portugal, 30 August - 3 September 2005.

[3] Tevfik Bultan, Xiang Fu, and Jianwen Su. Analyzing conversations of web services. *IEEE Internet Computing*, 10(1):18–25, 2006.

[4] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2000.

[5] Jianchun Fan and Subbarao Kambhampati. A snapshot of public web services. *SIGMOD Record*, 34(1):24–32, 2005.

[6] Howard Foster, Sebastián Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. In *Proceedings of 18th IEEE International Conference on Automated Software Engineering (ASE 2003)*, pages 152–163, Montreal, Canada, 6-10 October 2003.

[7] Howard Foster, Sebastián Uchitel, Jeff Magee, and Jeff Kramer. Tool support for model-based engineering of web service compositions. In *Proceedings of IEEE International Conference on Web Services (ICWS 2005)*, pages 95–102, Orlando, FL, USA, 11-15 July 2005.

[8] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting bpel web services. In *Proceedings of the 13th international conference on World Wide Web (WWW 2004)*, pages 621–630, New York, NY, USA, May 17-20 2004.

[9] Xiang Fu, Tevfik Bultan, and Jianwen Su. Synchronizability of conversations among web services. *IEEE Transactions on Software Engineering*, 31(12):1042–1055, 2005.

[10] Elsa L. Gunter and Doron Peled. Model checking, testing and verification working together. *Formal Aspects of Computing*, 17(2):201–221, 2005.

[11] Klaus Havelund, Michael R. Lowry, and John Penix. Formal analysis of a space-craft controller using spin. *IEEE Trans. Software Eng.*, 27(8):749–765, 2001.

[12] Gerald J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[13] Hai Huang, Wei-Tek Tsai, Raymond Paul, and Yinong Chen. Automated model checking and testing for composite web services. In *Proceedings of 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, pages 300–307, Seattle, WA, USA, 18-20 May 2005.

[14] IBM. Business process execution language for web services version 1.1. On-line. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[15] IDC. IDC research reports. On-line. http://www.idc.com/

[16] Neal Leavitt. Are web services finally ready to deliver? *IEEE Computer*, 37(11):14–18, 2004.

[17] Suet Chun Lee and Jeff Offutt. Generating test cases for xml-based web component interactions using mutation analysis. In *Proceedings of 12th International Symposium on Software Reliability Engineering (ISSRE 2001)*, pages 200–209, Hong Kong, China, 27-30 November 2001.

[18] Barbara Staudt Lerner. Verifying process models built using parameterized state machines. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004)*, pages 274–284, Boston, Massachusetts,USA, July 11-14 2004.

[19] Srini Narayanan and Sheila A. McIlraith. Analysis and simulation of web services. *Computer Networks*, 42(5):675–693, 2003.

[20] Jeff Offutt, Shaoying Liu, Aynur Abdurazik, and Paul Ammann. Generating test data from state-based specifications. *The Journal of Software Testing, Verification and Reliability*, 13(1):25–53, 2003.

[21] Jeff Offutt and Wuzhi Xu. Generating test cases for web services using data perturbation. *ACM SIGSOFT Software Engineering Notes*, 29(5):1–10, 2004.

[22] Organization for the Advancement of Structured Information Standards (OASIS). *ebXML*.

[23] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WSBPEL)*.

[24] P.Ammann, P.E.Black, and W.Majurski. Using model checking to generate tests from specifications. In *Procceedings of 2nd IEEE International Conference on Formal Engineering Methods*, pages 46–56, Brisbane, Australia, 1998.

[25] A.W. Roscoe and Philippa J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7(1):147–190, 1999.

[26] W3C. *Extensible Markup Language (XML) 1.0 (Third Edition)*.

[27] Jia Zhang. Trustworthy web services: Actions for now. *IEEE IT Pro*, pages 32–36, January/February 2005.

Initial Investigations into Interoperability Testing of Web Services from their Specification Using the Unified Modelling Language

Colin Smythe [1]

eLoki Limited
Sheffield, UK

Abstract

The latest paradigm for the development of distributed systems is the Service-oriented Architecture (SoA) with implementation based upon Web Services. In principle, the development of a distributed system using a model driven approach enables the automated implementation and testing of the system from the specification of the component services. An evaluation of this approach based upon the work completed as part of the Telcert Project funded under EU 6th Framework IST Programme is described in this paper. This work demonstrates that it is possible to auto-generate interoperability conformance tests for a service that has been described using a new SoA-Profile of the Unified Modelling Language. The generation of the interoperability tests is based upon a series of transformations of the XML Metadata Interchange (XMI) representation of the UML description through a new intermediate representation that provides tool independence and enables a richer description of the service. This approach is illustrated using an example service specification to explain how the set of tools produce the set of interoperability conformance tests ready for submission to a test system.

Key words: Web service, interoperability testing, model driven architecture, specification

[1] Email: colin@dunelm.com

1 Introduction

The Internet Engineering Task Force (IETF) was the original developer of open specifications and standards as a way of significantly improving system interoperability. This approach has been replicated by many other organizations, including the World Wide Web Consortium (W3C), with the latest emphasis being on Service-oriented Architectures (SoA) and their realization using Web Services. The success of this approach has spawned many systems, tools and applications for which interoperability conformance to one or more specifications/standards is claimed. However, while the importance of proving such claims is known to be important there has been limited success. Successful conformance testing has only been achieved where there is a clear commercial exploitation model e.g. Unix and "The Open Group"[2].

The IMS Global learning Consortium (IMS/GLC) is developing SoA-based open specifications for e-learning interoperability. The development of the corresponding interoperability conformance tests, while recognised as essential because of the broad adoption of the IMS/GLC specifications and the many claims of conformance, has proved problematic due to the high costs per specification. Therefore, there is an important need to develop simple and cost-effective interoperability conformance testing. It is even more appealing if this approach can be combined with the development of the specification as opposed to independently developed after the specification has been completed.

Specifications are traditionally word-processed documents with diagrams and tables used to describe data structures, their relationships and the associated behavioural constraints. The core specification is called the Information Model and this is expressed in a binding language neutral form. A further document is created that maps the Information Model to a language specific form, called the Binding document. The Extensible Mark-up Language (XML) is the current preferred language binding form for data models. In many cases a third document is created called the Best Practices document that addresses implementation and usage issues.

The work described in this paper investigates the issues of automatically generating the interoperability conformance tests from the computer-based representation of the specification against which conformance is claimed. The corresponding methodology is based upon the Model Driven Architecture (MDA) [4] approach and the usage of the Unified Modelling Language (UML) [14]. The specification of the service under development is described using a new profile of UML. The service is realized as a Web Service with a binding expressed in the Web Services Description Language (WSDL) and XML. The computer representation of this UML, one flavour or other of the XML Metadata Interchange (XMI), is then used as the source for a series of transformations that are used to produce a new representation of the specification and the associated WSDL/XML binding(s). Also, a separate set of tools use the new intermediate representation as the source for the generation of the interoperability conformance tests.

The origins of this work are from the IMS/GLC [2] and the Telcert Project [3]. The relevant IMS/GLC work is that of the IMS General Web Services (GWS) Base Profile specification [7] and the accompanying WSDL binding guidelines [14]. The IMS GWS specification defines a common approach to using Web Services as the realization of e-learning interoperability specifications based upon a SoA framework called the IMS Abstract Framework [10]. The IMS GWS consists of the Base Profile plus several extension profiles [5, 6, 8]. The IMS GWS Base Profile is derived from the Web Services Interoperability Organization's (WS-I) Basic Profile v1.1 [13, 15]. The IMS/GLC also has the IMS Binding Auto-generation Toolkit (I-BAT) that is used to automate the generation of the service and data bindings from the corresponding Information Model of the specification [14]. The usage of the I-BAT enables the Information Model and the Binding document and the binding validation files to be derived from the UML description i.e. a single point of entry is used as the source for all three sets of files. This removes the inconsistencies between these documents that are inevitable when the same information is entered using separate processes (as is normally the case).

Telcert (Technology Enhanced Learning: Conformance – European Requirements & Testing) was a research and innovation project under the European Union's 6th Framework programme. The aim of the project was to help transform the adoption of standards-based e-Learning products and services by simplifying the process of conformance testing. Therefore, the approach was to automate, as much as possible, the process of conformance testing by deriving and applying these tests directly from the specification(s) to which a vendor purports compliance for their tool, application, system or content. While the work on the Telcert project took an approach similar to that of the IMS/GLC there were significant differences:

a) Telcert was focused on how to use the original specification to generate the corresponding interoperability conformance tests. This requires the addition of extra information into the original specification to aid the automated test generation;

b) Telcert addressed the issue of interoperability conformance testing against an application profile of a specification and not just the original specification. It is accepted practice within e-learning to profile specifications, to meet a specific set of requirements e.g. SCORM2004 [1], etc. Therefore, all of the techniques and tools developed under Telcert are applicable to profiles as well as the original specification;

c) Telcert developed an approach that was independent of the tools used to create the computer-based representation of the Information Model of the specification. Early experience showed that the UML tools used to create the representation of a specification did not provide interoperability of the representation. Therefore, Telcert developed an approach that minimised the effects of these tools differences.

[2] http://www.imsglobal.org
[3] http://www.opengroup.org/telcert/

It is important to stress that the work described herein is not limited to use in e-learning. While e-learning has been used to prove and demonstrate the concepts and methodology, the approach has been developed such that it can be applied to any specifications that are produced using the methodology and its supporting tools. Therefore, this approach can be used to provide conformance testing of any system, tool or application for which conformance, to a SoA-based specification that has been developed using this methodology, is claimed.

The rest of this paper presents the issues that need to be addressed when testing SoAs and their realization using Web Services (Section 2); describes this new methodology in terms of the specification definition to test generation workflow (Section 3); demonstrates the specification of a Web Service using the methodology (Section 4); describes the set of interoperability conformance tests that can be computer-generated from the specification (Section 5); and analyses the strengths and weaknesses of the methodology (Section 6).

2 Testing Service-oriented Architectures & Web Services

2.1 Service-oriented Architectures and Web Services

One of the attractions of SoAs is that a system can be constructed from a collection of component services. The functionality of each service is revealed by the external interface to the service. In the case of Web Services, this interface is defined by the WSDL file. As shown in Figure 1, a SoA consists of "Service Providers" (servers) and "Service Requesters" (clients). The service is defined using an abstract service interface, which is realised using an appropriate binding and implementation technology: in programming terms this interface is revealed to the end-system applications as an Application Programming Interface (API). The abstract service interface is defined during the specification process and it must include the definition of the required end-system behaviour to ensure that the required sequence of events can be supported. When the service is realised using Web Services, the abstract service interface is realized as the exchange of a sequence of messages (in the case of a Web Service, these could be SOAP messages) that are used to carry the information between the client and server (in Figure 1 the messages sent between "A" and "B"). From the perspective of an interoperability specification, the behaviour of the end-systems are only defined in terms of the information exchanged i.e. the internal operation of an end-system can only be implied by the information received from it.

2.2 Interoperability Testing of a Web Service

From the perspective of interoperability testing, the testing of a service can only use the functionality supported by the service i.e. only those operations defined for the service can be used for testing the service. Consider the case of a Web Service as shown in Figure 2. The Web Service is assumed to be XML data, carried in SOAP messages across an HTTP web server infrastructure. If the service is defined to

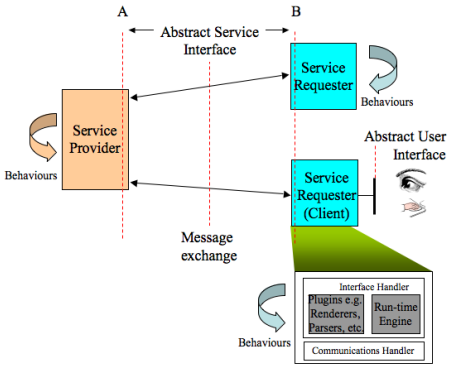


Fig. 1. A schematic view of the features of a service-oriented architecture.

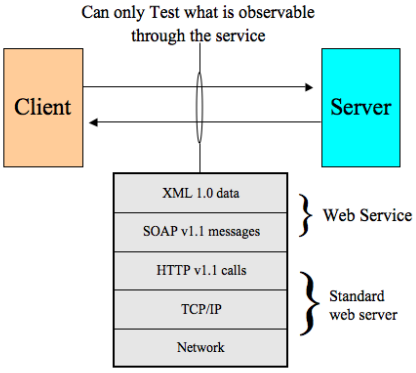


Fig. 2. The limitations of interoperability testing of a Web Service.

have the operations "createObject", "readObject" and "writeObject" (where "Object" could be any form of XML data structure) then interoperability testing must ensure that a client or server implementing this service should test for each of these operations and for any valid and invalid sequence of operations. Also, in principle, the testing itself can only use those three operations (in this example this means that it is not possible to delete an object once it has been created). The further constraint is that it is not possible to observe the internal behaviour or state of any internal data structures except by implication i.e. data persistence can be implied by a create operation followed by sequential reads of the same object.

Hardware testing has similar constraints and these are overcome by creating a separate test interface which when used in conjunction with the interface under test reveals information not available through the interface under test. This special test interface may or may not be made available as part of the commercially distributed equipment. The same approach can be adopted for testing SoA. A separate service test interface can be defined and used solely for testing of the service. This means that interoperability testing can include features beyond the specification of the service i.e. internal observation of a data structure, etc.

2.3 Simplifying the Service Description

Most interoperable Web Service implementations are relatively simply due to the immaturity of the standards and development tools. For this work the functionality of the service under specification has been constrained (these constraints do not invalidate the approach but they do focus the evaluation on the key issues):

- Service description using a new SoA-Profile of UML – the specification is developed using the Telcert SoA-Profile (see Section 4) and the corresponding Telcert XMI format[3]. There is nothing unique in the Telcert UML Profile that could not be achieved using another equivalent UML SoA-Profile. However, the Telcert tools are designed to use the Telcert SoA-Profile;

- Synchronous messaging – there are many different message choreographies that can be supported between a client/server for a single service. The simplest reliable client/server choreography is "Synchronous" in which the client issues a "Request" message and the server replies with a "Response" message. Every "Request" must have a "Response" and only one "Request" can be outstanding i.e. the client processing is blocked until the "Response" message has been received;
- Single request processing – a server is capable of receiving many concurrent responses i.e. in general there will be several clients interacting with the server at any one moment in time. For this analysis, only conformance tests of the server operating on one request at a time are considered. Therefore, testing issues such as concurrency and the performance capability of the server are not addressed. Concurrent data access is an important aspect of any server but conformance testing of this requires that this functionality is expressed in the specification itself. This is possible in UML but adds considerable complexity to the analysis of the protocol state model;
- Stateless service – traditional HTTP interactions are stateless. The inclusion of "cookies" is one method of supplying state awareness, however, the usage of "cookies" create security vulnerabilities. Therefore, this analysis assumes that the service is stateless i.e. each operation request is self-contained and does not require a previous operation to have been undertaken. Note that it is the interaction that is assumed stateless. Both the client and server are expected to be stateful e.g. the server is expected to persist any data written to it unless explicitly deleted;
- Single service – it is increasingly common practice to create Web Services that are functionally cohesive and then to create more complex Web Services by chaining the smaller ones together. This involves choreographing these services and languages such as Business Process Execution Language for Web Services (BEPL4WS) are being developed for this purpose. For this analysis the focus is on the specification of a single service;
- Predefined status reporting mechanism – it is assumed that each operation is terminated at the client with the reporting of the appropriate status information. This means that the response messages must contain a status information reporting structure. In general this structure will vary from service to service, particularly if the services are defined by different organizations. This work has adopted the IMS GWS status reporting structure [14]. This information is carried as an IMS SOAP message header extension;
- Testing of the server – the focus is on the testing of the remote server through its web service interface. This is because it is easier to instigate the tests directly for the server than for the client. Server testing requires a shell client to be created that is capable of constructing the relevant web service call in the form of the request SOAP messages. This form of testing means that the actual server is used i.e. the server can be tested in its operational environment; only the server URL

is required. In the case of the client, specialist user interface testing software is required and must be configured/trained to drive the user interface correctly to cause the subsequent creation of the corresponding web service calls;

- Web service profile – the Web Service profile being used is based upon the IMS GWS[7]. This states that the Web Services profile is XML v1.0, with SOAP v1.1, with HTTP v1.1 and that the Web Service must be expressed using WSDL v1.1;
- Baseline Tools – Poseidon v3.2 as the UML authoring tool and Oxygen v7.0 as the XSL/XML/XSD authoring tool. The demonstration of the principles is subject to the capabilities of specific tools but the concepts can be easily realised in equivalent tools.

3 Interoperability Testing using a Model Driven Approach

3.1 The Workflow

Organizations that develop specifications or standards for services use different documentation techniques and tools. The documentation covers the same material i.e. the Information Model and the Binding Definition. The binding should be an accurate expression of the Information Model in the binding language but this may not always be possible i.e. the syntax and semantics of the Information Model and binding language may not completely compatible. If there are compatibility problems it is essential that the incompatibilities be treated consistently otherwise implementation problems will become more likely. Therefore, there an organization must use a specification development workflow that addresses all of these issues.

The process for developing a testable specification based upon UML is shown in Figure 3. The process goes from left to right, such that:

- The initial stage is to create the UML description of the specification. There are several UML authoring tools available each of which stores their files using the XMI format. Unfortunately these tools use XMI in non-interoperable ways and so it is not possible to import a description produced by one tool into another. The problem is further compounded by the fact that the description of a specification may require features that are unsupported by the UML tools e.g. Protocol State Machines (PSMs). The solution is to create an intermediate representation of the specification to replace the XMI;
- Once the specification has been created, its description is transformed into the intermediate representation. Therefore, the latter stages of the process use the same file format and content structure. This transformation is achieved using an XML Stylesheet (XSL). There is a different XSL for each authoring tool and this transformation creates the new format of the specification with a “txmi” file extension;
- The format of the “txmi” is validated using the “servicexmi.xsd” file. This XML

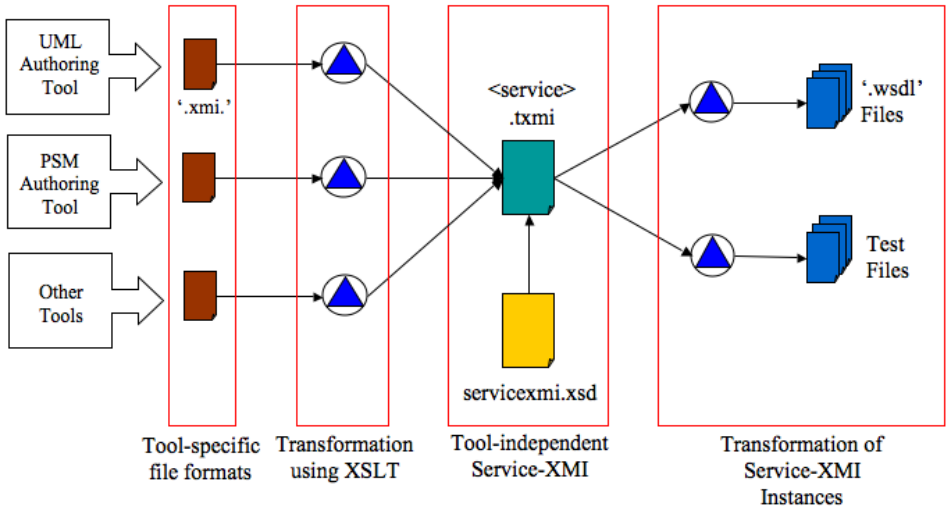


Fig. 3. Workflow for specification-based testing of a web service.

Schema Definition (XSD) is the control document for all "txmi" instances. The structure of this XSD is such that it ensures that all of the required information for producing a testable specification has been supplied. It is also possible to edit the "txmi" instance to add other information e.g. to refine the transition descriptions in the protocol state machine. These edits can be supported using a standard XML authoring tool;

- The final stage is to use the "txmi" instance to create the corresponding binding and tests for the specification. The WSDL/XSD binding files are created using XSLs whereas the test files are created using purpose built software. Both the binding and test files are used by the test system. A Java application has been developed that is used to evaluate the capability of simple test generation algorithms. This application does not "walk" the PSM, instead it uses the characteristics of the conformance statement and the PSM transitions to investigate how much of the test coverage can be generated with very simple analysis.

It is important to note that the approach shown in Figure 3 is applicable to both the development of the original service specification or a profiled version of the specification. The profiled version is produced using the UML–authoring tool to change, appropriately, the UML description of the base specification. The transformation tools are then applied to the UML representation of the profiled specification and not that of the base specification. Further information on the profiling of data models is available in [16, 17].

3.2 Service Testing

Once the base or profiled specification has been used to generate the set of conformance tests, the final stage is to test the implementation(s). A schematic representation of the required test architecture is shown in Figure 4. The salient features of the test architecture in Figure 4 are:

- Test database – this is the database of the full set of tests that are to be used. This database contains the set of tests expressed in the test system language that is interpreted by the server and client test harnesses. This database is also the store for the messages from the implementations under test;
- Test script – this is the set of test scripts that must be invoked to test a client. These test scripts contain the instructions that must be followed to drive the client interface in the correct manner to achieve a particular output web service call. These scripts are specific to the client interface (at present this form of testing is not being considered);
- Implementations under test (service provider and service requester) – these are the implementations that are to be tested. An implementation under test would be either a server or a client but the test system must be capable of testing both;
- Test service requester – this is the shell client that is used to test the conformance of a server implementation that is being tested. This shell client can send any form of request message and can receive any form of response message. The response is returned to the test database to be analysed once the set of tests have been completed;
- Test service provider – this is the shell server that is used to test the conformance of a client implementation that is being tested. This shell server can send various different responses for each request message (the exact response is determined by the test scripts contained in the test database and these scripts have been generated from the UML-based specification).

The current focus is on server-side testing only. In Figure 4 this is covered by the top set of boxes i.e. the "Service Provider" being tested by the "Test Service Requester". This means that a real server can be tested using the common shell client as driven by the test database.

3.3 UML Representation of the Specification

UML is used as the description representation of the Information Model for the specification under development. UML is a rich modelling language and so its usage must be defined and constrained to ensure a consistent approach to the development of different services. This is achieved by using an appropriate profile of UML. A different profile is defined for each workflow and the usage of the workflow i.e. the development of tests from the specification will require a different, but in general related, profile from that which is required just to describe the same specification (the actual profile used by Telcert is described in Section 4).

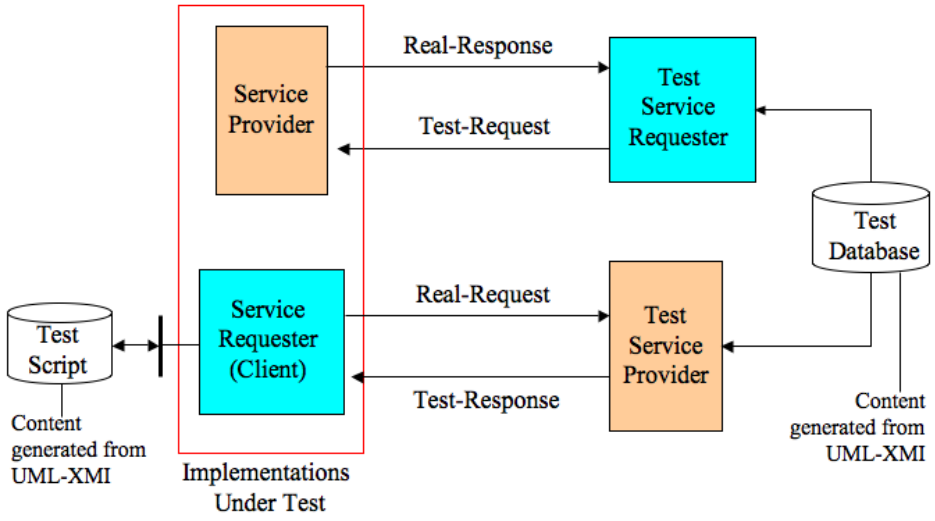


Fig. 4. Schematic representation of the test architecture for conformance testing.

There are many specification languages and techniques that could be used for this type of work. Formal techniques such as "Z" and its various flavours are possible, less formal approaches such as Petri Nets and finite state machines or standard software development techniques such as UML are also available. UML was selected because:

- A lot of MDA work has been driven by the ideas considered during the development of UML;
- There are many UML authoring tools available for different platforms and so UML is available to most specification engineers. This is not true for techniques such as "Z" or Petri Nets;
- UML has broad adoption, increasing daily, within the software engineering community and so many specification development engineers are already familiar with its syntax and semantics;
- UML file formats have an agreed representation form, XMI, even though there are many implementations problems that cause interoperability problems. The usage of XMI means that it is relatively simple to transform the UML representation into other forms.

It is important to note that the usage of UML is a "means to an end". The automated creation of conformance tests directly from the specification is the objective of this work. If the approach can be shown to work for one specification representation language then other languages can be investigated at a later date. Therefore the initial selection was required to simplify the investigation process as a whole.

4 The Specification of Web Services

4.1 UML Profiling for a Web Service Description

UML is an extensive language with features that are useful at most stages of the development and implementation of a system [14]. UML 2 has been extended with features that enable it to be tailored for a specific purpose e.g. the development of source code using a particular development environment. These features also enable the language to be used to support automated transformations i.e. the transformation of a specification into a particular implementation language e.g. Java. A UML SoA-Profile has been created [3] that enables the development of a testable service specification that can be realised in more than one binding (in this work the actual bindings are limited to XML, SOAP and WSDL). This UML SoA-Profile is based upon tailoring UML with the following features (these features are not unique to the requirements of XML, SOAP, WSDL, etc. but do provide coverage of their syntax and semantics):

- Package – these are used as the primary container for each of the specification description diagrams;
- Stereotypes – these are used to identify specific types of a UML feature e.g. a set of types of Packages, a set of types of classes, etc;
- Data-types – the built-in data-types that are to be used when creating the service and data models. Only these data-types are permitted in a specification;
- Comments and documentation – these enable comments and documentation features to be added to the description and supplied as comments within the binding files;
- Object Constraint Language (OCL) usage – OCL is used to express the preconditions and postconditions used in the protocol state machine description for transitions.

These features are designed to use the functionality supported by the majority of UML authoring tools. The corresponding templates have to be created for the UML authoring tool to be used for the specification.

4.2 The SoA UML Profile for a Web Service Description

Within UML, “Packages” are used to organize descriptions in logical collections. Packages can contain Packages and they can be linked to each other to show their dependencies. For the SoA-Profile, Packages are used as the logical container for the different parts of the specification. This ensures that the descriptions of the model are isolated from any of the tool-specific descriptions e.g. diagram layouts, etc, and therefore the transformations also use the “Package” description as the context. The set of SoA-Profile Packages and their usage is listed in Table 1 (the bracketed terms under the Package name denote the multiplicity in the service description).

Package	Description
Conformance [0..*]	Each Service must have a corresponding Conformance Statement. Each Conformance Statement is contained within its own Conformance Package. This statement is used to define which parts of the service an implementation supports.
DataModel [1..*]	A data model is described using a set of class descriptions. These descriptions must be contained in one or more DataModel packages. At the current time the distribution of the classes amongst the DataModel Packages is not significant i.e. the distribution should be used to aid readability of the specification.
DataTypes [1]	Within the "Telcert" Package this is the container for all of the in-built primitive and derived data-types. A derived data-type constrains a primitive data-type. These data-types replace all of the in-built types defined by the UML tool.
Preferences [1]	Contains all of the preferences and default values that are to be used by the binding transformation algorithms.
ServiceGroupModel [1]	Each specification must have one and only one ServiceGroupModel Package. This Package is used to contain the binding and documentation context for the entire set of service descriptions.
ServiceModel [1..*]	Each specification must have at least one ServiceModel Package. This Package is used to contain the description of the service in terms of its interfaces.
TestModel [0..*]	The set of classes that are added to support the precondition and postcondition statements made in the protocol state models. These classes are not a part of the data exchanged by the service.

Table 1
The set of UML packages in the service-Profile.

It should be noted that a PSM description is NOT contained within a Package. The PSM is linked to a particular interface for a particular service (more than one PSM can be associated to the same interface and service combination). The definition of the associated control XSD for the validation of the SoA-Profile instances (the "servicexmi.xsd" in Figure 3) is available in [3]. Therefore, using these Packages, each service specification is undertaken by:

a) The specification file is created using an instance template of the UML SoA-Profile. This template contains all of the steroetypes, data-types and diagram outlines which have been defined in the template for the particular UML authoring tool to be used;

b) The "ServiceGroupModel" description is produced. This description identifies the target bindings for the set of services e.g. the way in which the Web Service is to be expressed in WSDL;

c) The "ServiceModel" for each of the services to be described in the specification is produced. Whilst it is possible to define more than one service in the specification it is recommended that only one service per specification file be described. The "ServiceModel" contains one or more stereotyped interface class each of which contains one or more operations. There is no description of the messaging model e.g. using sequence diagrams. The messaging exchange choreography is create by the binding transformation tool;

d) The set of "DataModels" are now created. It is recommended that each Data-Model Package be used to collect a set of common classes i.e. those that are functionally related. The description must also include the status information reporting classes and the SOAP header structures for the request and response messages. Every complex parameter in the ServiceModels must also be described in the DataModels. Note that the SoA-Profile for the data models dopes not assume that XML is the target binding language. Therefore the SoA-Profile is very different to the equivalent attempt to represent XML using UML [12].
Other profiles provide stereotypes that reflect the syntax and semantics for creating XML. However, this binds the Information Model directly to XML and this can be problematic when using other binding languages.

From the point of view of conformance testing the following additions are now made:

a) The set of PSM models must now be created. Each of these PSMs must be linked to a service and a particular interface within the service. The PSM consists of a set of states that are interconnected using transitions. Each transition must consist of the preconditions and postconditions as expressed in OCL;

b) The "TestModels" are now created. The "TestModel" Packages contain all of the special classes that are used in the precondition and postcondition statements within the PSM models but which are not defined in the "DataModel" Packages;

c) Finally the set of Conformance Statements are created. There is a separate Conformance Statement Package for each Service and in each there is a statement from the perspective of the clients and/or server. These statements define the implementation support or otherwise for each operation in the interface.

4.3 The Service Model

Consider the specification of a Content Repository Access Service (CRAS). What is stored in the repository is not the focus of this example. The CRAS client/server

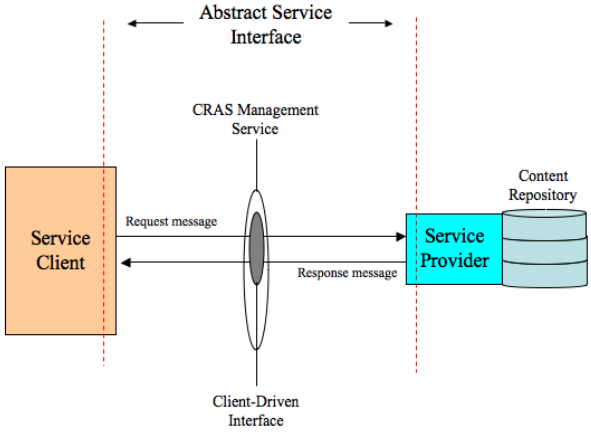


Fig. 5. The Content repository access service architecture.

architecture is shown in Figure 5; the CRAS is the interface between the client and the server (cf. Figure 1). As shown in Figure 5, access to the content repository is via the CRAS management service that consists of one or more interfaces.

The only interface defined in this current version is that called the "Client-Driven Interface". This interface describes the operations that can be invoked by the client. In this version of CRAS the server cannot initiate operations. This service is to be implemented as a Web Service using SOAPv1.1 with Message Transmission Optimisation Mechanism (MTOM) [6] (this is an extension of SOAP with attachments and enables non-XML data to be included in the SOAP messages). The binding will be described using WSDLv1.1. The binding also assumes synchronous communications i.e. once the client has issued the "request message" then it is blocked until the server "response message" is received. This means that only one request at a time can be issued. The service model is now produced (the definition of the ServiceGroupModel is not presented) as shown in Figure 6 and Table 2 (the full description of the CRAS is given in [11]).

The salient features of the service model description are:

- The service name is "CRASmanagement". The "Legend" information for the service is supplied to provide version control, dating, etc.
- The interface "ClientDriven" contains the four operations of createPackage, deletePackage, readPackage and updatePackage (see Table reftab:tab2 for the details of these operations). The corresponding comments are also supplied (these comments will be inserted in the WSDL file as part of the PortType description). Each operation has a "return" parameter that has the class "StatusInfo";
- The "Binding" is for SOAP v1.1 that defines the usage of MTOM as the SOAP message attachment mechanism;
- The set of class stereotypes are used by the transformation XSLs to correctly interpret the SoA-Profile.

Operation	Description
createPackage	To request the creation of a populated “Package” on the target system where the source is responsible for the allocation of the unique identifier.
deletePackage	To request the deletion of a “Package”. The “Package” record is deleted and all of its associated resources.
readPackage	To read the full contents of the identified “Package”. The target must return all of the data it has for the identified “Package”.
updatePackage	To replace the content of the identified “Package”. The target must write the new data into the “Package” record. This is a destructive write-over of all of the original information.

Table 2
Summary of operations for Client-Driven interface in the CRAS management service.

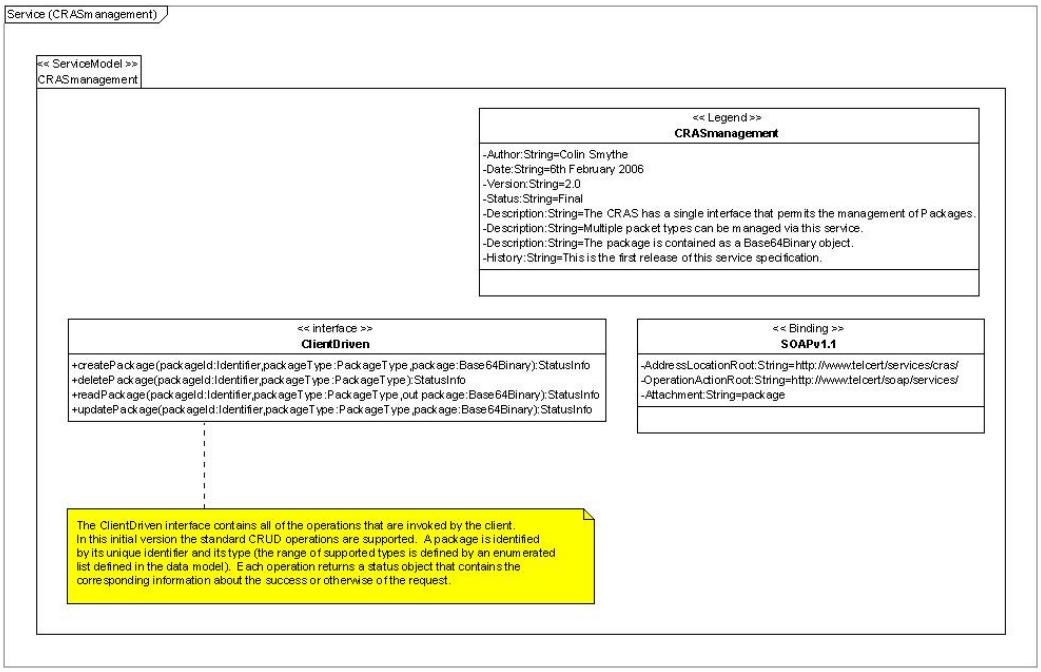


Fig. 6. The ServiceModel description of the CRAS using the UML profile.

4.4 The Data Model

The CRAS data model is based upon two types of description:

- Common service features – the data objects that are used to construct the common service features i.e. the parameters exchanged for the operations, the status objects, etc.

- Package specific descriptions – each type of package has its own separate UML data model description. The package is exchanged as a message attachment and so it is the implementation that is responsible for using the relevant data model for type of package being processed. The data models for the package are not included because from the perspective of the service, this information is treated as Base64Binary encoded data in the SOAP message. The data model description is given in more detail in [3].

The common service data model features for the CRAS are shown in Figure 7. The salient features of this data model package shown in Figure 7 are:

a) The "StatusInfo" class is used to describe the status information returned for an operation. There are three codes sets associated within this object
   · CodeMajor – the top-level status code classification of "Success", "Processing", "Failure" and "Unsupported"
   · Severity – identifies the classification of the coding i.e. "Status", "Warning" or "Error"
   · The CodeMajor/Severity status information matrix is derived from the IMS GWS documents [7, 14]
   · CodeMinorFieldValue – this is the set of business specific status codes;

b) The set of packages types that are supported are defined by the list in the class "PackageType". Seven types of package are currently defined, one of which is a generic "other" i.e. "assessment", "content", etc. (recall that the UML descriptions of these packages are not presented in this paper);

c) The "StatusBinding" class and sub-classes are used to support the definition of the messages (these are predefined and linked explicitly to the binding transform XSLs).

4.5 The Protocol State Machine Model

The full protocol state machine for the "ClientDriven" interface of the CRASmanagement service is beyond the scope of this paper. A subset of the protocol state machine is shown in Figure 8. The manner in which the protocol state machine is described and the format for the statements assigned to the transitions are tool dependent.

The states are linked by transitions and these transitions describe the events that cause the change of state. Each transition becomes a row in the equivalent state table for Figure 8 and each transition consists of one permissible value for the status information (StatusInfo) parameter. The state table row descriptions are composed of:

a) Current State – the state from which the transition is triggered;

b) Precondition – this is the list of conditions that are true in the system for the operation to be triggered. These conditions are expressed using OCL and expressions must be established for all of the input parameters defined in the ServiceModel (see Figure 6). Other system conditions will be defined to constrain when the

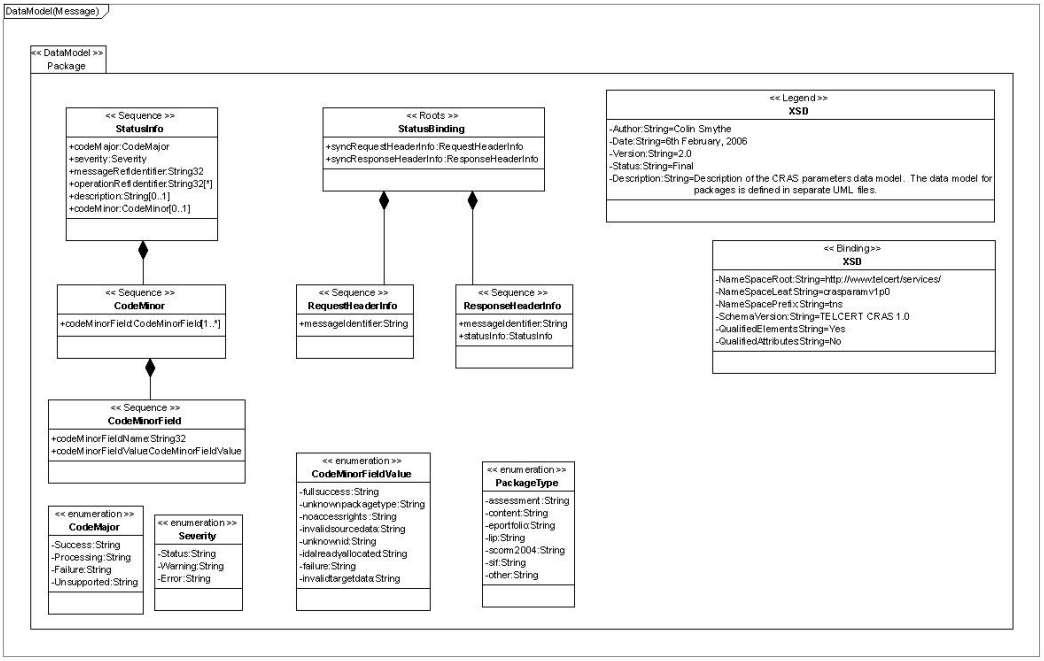


Fig. 7. CRAS common features data model description using the UML profile.

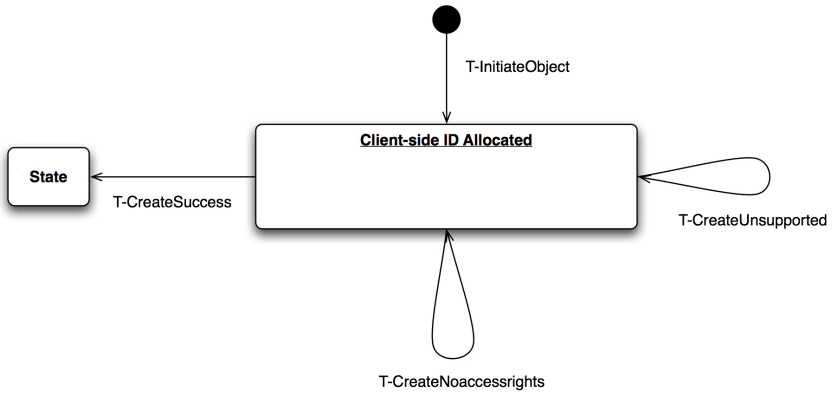


Fig. 8. Protocol state machine describing the CRAS.

transition can take place and to ensure that the required status information is reported;

c) Message Name (Parameter List) – this is the name of the operation that causes the transition. The set of parameters are as defined in the interface representation;

d) Postcondition – this is the list of conditions that are true in the system when the operation is completed. These conditions are expressed using OCL and expressions must be established for all the output parameters defined in the Service-Model (see Figure 6). This list of conditions must include “StatusInfo.CodeMajor”, “StatusInfo.Severity” and “StatusInfo.CodeMinor”. Other system conditions will be defined to constrain the state of the system once the transition has completed;

e) Next State – the state for the system when the transition has completed.

State		Transition	Comments
Current State	Next State		
CreatePackage			
Client-side ID allocated	State	*Label:* T-CreateSuccess *Message:* createPackage (packageId, packageType, package) *Preconditions:* • PackageIdClientSet → includes(packageId) • PackageIdServerSet → excludes(packageId) • PackageTypeClientSet → includes(packageType) • PackageTypeServerSet → includes(packageType) • ValidPackageSet -> includes(package) *Postconditions:* • result.StatusInfo.CodeMajor=Success • result.StatusInfo.Severity=Status • result.StatusInfo.CodeMinor=fullsuccess • PackageIdClientSet → includes(packageId) • PackageIdServerSet → includes(packageId) • PackageTypeClientSet → includes(packageType) • PackageTypeServerSet → includes(packageType) • RepositorySet = RepositorySet@pre → including(package)	A successful creation of the repository. A new package is created on the server. All of the parameters are valid and a valid package is supplied. The parameters are: • packageId – unique identifier for the package • packageType – the type of package being created • package – the package itself.

Table 3
A partial state table for the protocol state machine.

A partial state table for the PSM shown in Figure 8 is listed in Table 3.

The set of precondition and postcondition statements are expressed in OCL. In Table 3, the set of preconditions define the fact that the identifier assigned to the package must not be allocated on the server [PackageIdServerSet -> excludes(packageId)] and that the package type must be valid for both the client and server [PackageTypeClientSet -> includes(packageType, PackageTypeServerSet -> includes(packageType]. The set of postconditions now define the status information returned by the server [result.StatusInfo.CodeMajor = Success, result.StatusInfo.Severity = Status, result.StatusInfo.CodeMinor = fullsucces]. The final postciondition states that the package must be stored in the repository [RepositorySet = RepositorySet@pre ->including(package)].

Most UML authoring tools enable languages other than OCL to be used in expressing the conditions. OCL has been used because it is the preferred language in UML [14]. There are problems and limitations with OCL and a language such as "Z" may be better suited to the specification needs. The other issue is that many

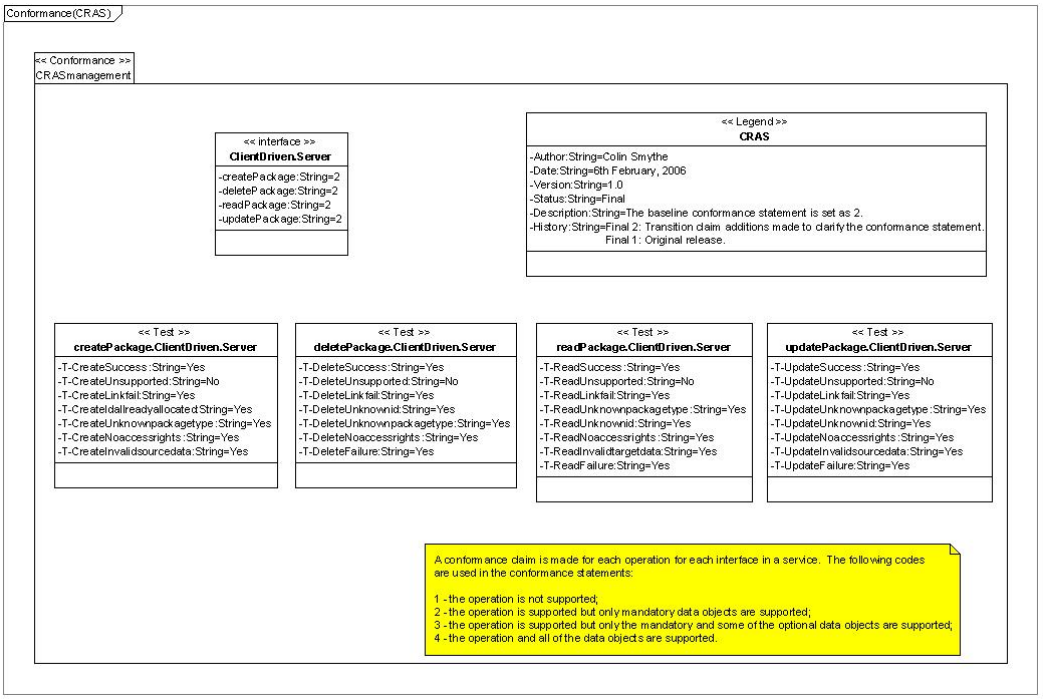


Fig. 9. The conformance statement for the CRAS using the UML profile.

UML authoring tools do not provide an OCL parser and thus they store the OCL expressions as strings. This means that it is possible to enter syntactically and semantically invalid OCL expressions.

4.6 The Conformance Statement

The Conformance Statement is shown in Figure 9. This conformance requirement is the minimum level that ensures that all of the operations are supported.

This conformance requirement is assigned to the service (this should be one of the services defined by a service model) through the name of the Conformance package and the interface is identified using the "interface" stereotyped class name (this should be one of the interfaces defined for the service). The full conformance model will consist of one statement per service and each conformance package will have a conformance interface class for each of the interface classes in the corresponding service model. The "interface" class identifies the extent to which each operation is supported. For each operation that is supported there is a corresponding statement of the transitions that are supported for that interface. These transitions correspond to those marked in the PSM (this means that while an operation may be supported not all of its functionality may be available).

In Figure 9 the test class "deletePackage.ClientDriven.Server" shows that the transitions "DeleteSuccess", "DeleteLinkfail", "DeleteUnknownid", "DeleteUnknownpackagetype", "DeleteNoaccessrights" and "DeleteFailure" are supported by the server under test. All of the transitions named in the "deletePackage.ClientDriven.Server" test class must be labelled in the PSM and are the transitions that

001	**Main()**	The main algorithm.
002		
003	**for each** Service **do**	Each 'ServiceModel' is analysed.
004	**begin**	
005	**for each** Interface **do**	Each 'Interface' Class in the 'ServiceModel' is analysed.
006	**begin**	
007	**for each** Operation **do**	Each Operation in the 'Interface' Class is analysed.
008	**begin**	
009	**if** Operation 'Unsupported'	If the 'ConformanceStatement' states that the
010	**then** *UnsupportedOperationTest()*	Operation is not supported then this must still be
011	**elseif** Transition 'Unsupported'	tested. If the 'ConformanceStatement' states that
012	**then** *UnsupportedTransitionTest()*	the Transition is not supported then this condition
013	**else** *SupportedTransitionTests()*	should still be confirmed by testing. The set of
014	**end** (Operation loop)	tests, and the expected responses, for this
015	**end** (Interface loop)	Transition must now been generated. This is the
016	**end** (Serviceloop)	entry point for the algorithm when there is no
017	**MainEnd**	'ConformanceStatement'.

Table 4
The main algorithm for generating the interoperability test.

can occur as a result of the "deletePackage" operation for the service being invoked.

For a base specification, the conformance requirement includes all of the operations and support for each of the transitions. One of the activities undertaken during profiling is to change the conformance requirement, to reflect the profile. A realization of the specification would entail the corresponding conformance statement being produced from the conformance requirement i.e. identification of what the implementation to be tested supports. It is the conformance statement that is used as the basis for the generation of the conformance tests to check that the implementation does indeed reflect the conformance statement. It is a separate issue, currently out-of-scope of this work, to determine whether or not the conformance statement is an acceptable minimum coverage of the conformance requirement.

5 Computer Generation of the Interoperability Tests

The generation of the set of tests is based upon the contents of the service "txmi" instance file. The content of this file must be interpreted without dependence on any knowledge of the context of the specification or any explicit naming convention that has been adopted to aid human readability of the UML description. In this work there is no attempt to generate the full set of possible tests or to attempt category partitioning (this work is undertaken elsewhere [3]). Instead, the coverage that can be easily obtained using as simple a set of algorithms is investigated. The main algorithm is presented using the pseudo-code in Table 4.

The salient features of the algorithm in Table 4 are:

- Tests must be generated for each service (line 003) within which tests must be created according to the set of Interfaces (line 005) and the Operations (line 007). A "ConformanceStatement" for the service may or may not exist. If no such statement is given, tests for all of the interfaces and operations are generated;
- If an Operation is "Unsupported" then there may still be a need for a corresponding test e.g. a service call that results in an "unsupported" response message (line

001	**SupportedTransitionTests()**	Generating the set of tests for each transition.
002		
003	noofTests = **if** Operation Conformance = 'Mandatory'	The total number of tests is the product of the
004	**then** *mandatoryGenKeyTestStateValues()*	number of test states that need to be evaluated for
005	**elseif** Operation Conformance = 'Optional'	each parameter in the Operation. This depends
006	**then** *optionalGenKeyTestStateValues()*	on the required completeness of the test coverage,
007	**else** *allGenKeyTestStateValues()*	the data coverage in the 'ConformanceStatement'
008		and the precondition and postconditions.
009	**for all** noofTests **do**	A separate test needs to be generated for each
010	**begin**	combination of the parameters conditions.
011	*getNextKeyTestStateValue(ParameterIn[1])*	To generate the test the value of each of the 'In'
012	…	parameters needs to be obtained.
013	*getNextKeyTestStateValue(ParameterIn[m])*	
014	*createTest(*	Generate the test using the appropriate test
015	*ParameterIn[1],*	representation language. This may or may not
016	*…,*	require knowledge of the binding form.
017	*ParameterIn[m]*	
018	*)*	
019		
020	*getNextKeyTestStateValue(ParameterOut[1])*	To generate the test the value of each of the 'Out'
021	…	and the 'Return' parameters needs to be obtained.
022	*getNextKeyTestStateValue(ParameterOut[m])*	
023	*getNextKeyTestStateValue(ParameterReturn)*	
024	*createTestResponse(*	Generate the expected test response using the
025	*ParameterOut[1],*	appropriate test representation language. This
026	*…,*	may or may not require knowledge of the binding
027	*ParameterOut[n],*	form.
028	*ParameterReturn*	
029	*)*	
030	**end** (noofTests)	
031	**FunctionEnd**	

Table 5
The algorithm for generating the interoperability tests for a specific transition.

010). Also, not every transition related to an operation may be supported but it may still be necessary to test for such transitions (line 012). Finally, the tests for the supported transitions are generated (line 013).

The algorithm for the generation of the tests for the supported transitions is listed in Table 5. The salient features of the algorithm in Table 5 are:

- The total number of tests that need to be generated depends on the nature of the "ConformanceStatement" i.e. the smallest number occurs when only the mandatory data structures are supported and the most being when all of the mandatory and optional data structures are supported (lines 003-007);
- The generation of the test state values (lines 004, 006 and 007) is determined by the set of preconditions and postconditions defined for the transition. These conditions (expressed using OCL) must define the range of permitted values for each parameter e.g. as min/max value. In this work the OCL is processed using string matching i.e. there is no formal parsing or analysis of the logic expressed in the OCL. The algorithm for generating the values is beyond the scope of this paper;
- Once all of the test conditions have been identified, the final step is to generate each of the individual tests (lines 011-018) and the corresponding expected response (lines 022-029). This generation requires the value of each parameter to be assigned to the appropriate structure in the binding format (or the intermediate test representation language format). The values for the tests are derived from

the set of "In" parameters and for the test responses they are derived from the set of "Out" and "Return" parameters. The tests must be expressed in the appropriate test representation language (for the Telcert project this is proprietary to the Open Group).

6 Strengths & Weaknesses of the Approach

The work completed thus far shows that the strengths of the approach are:

a) The usage of UML as the representation method for specifications and the subsequent electronic processing of the UML-XMI enables single entry multiple usage of the design information. Therefore the Information Model, Binding Document and the binding validation files can be consistently and repeatedly generated from a single representation source;

b) During the work, several new versions of the Poseidon UML authoring tool were released. Some versions contained changes to the way in which the UML description was stored in XMI. The changes to the XSL that transformed from the tool-specific XMI to the "txmi" instance format were relatively simple as the XSL was just a few hundred lines long. Alternative work on an XSL that transformed directly from the tool-specific XMI to the target binding was considerably more difficult as it was several thousand lines of code. Therefore, the usage of the new intermediate XMI is essential to simplify maintenance of the tools;

c) Several different services and data models have been generated using this methodology. There have been refinements to the approach e.g. the addition of new datatypes, refinement of the set of stereotypes, etc. but it has always been possible to realise the service binding as required by the engineer. Therefore it is reasonable to conclude that this approach is robust for the design of service and data models or that at the very least it is simple to extend to support any idiosyncratic aspects of a service or data model;

d) The methodology has an elegant mechanism for enabling profiles and implementation-specific information to be added. Profiling of the specification is achieved by creating a copy of the base specification UML description and amending it as required e.g. altering multiplicity values in the data model, adding new interfaces, etc. Implementation-specific information is added through the usage of the Conformance Statement Packages. These statements are used to tailor the set of tests that must be generated to show that the implementation meets its conformance claim;

e) While there are clear limitations on the types of tests that can be generated using the simple set of algorithms that are possible using string matching on the preconditions and postconditions, it is clear that a useful set of conformance tests can be readily generated.

The work completed thus far shows that the weaknesses of the approach are:

a) The message choreography for the service is expressed by rules built into the auto-generation tools. These rules should be removed and the message choreography expressed in UML as sequence diagrams. Sequence diagrams are used to express message exchange between two or more objects. Therefore, the server and client objects are defined and the service interaction is described using the "Request" and "Response" message exchange. These diagrams can then be used to express more complex message choreographies e.g. asynchronous communication with the usage of 2Request", "Response" and "Acknowledge" messages;

b) New classes in the DataModel Packages must be added so that the PSM transition descriptions are complete and consistent. These classes must be identified using a new stereotype that is used to inform the transformation tools that these classes are not to be included in the creation of the binding;

c) A more formal approach to the generation of the conformance tests requires that the PSM be "walked" i.e. each state must be reached through the appropriate sequence of operations. Sequence independent testing of each operation means that it is not possible to look at data persistence and consistency issues e.g. to ensure that once an object has been deleted that any identifier originally assigned can be reused;

d) There is a limit to the set of interoperability tests that can be generated from the description of the service alone. These limits include concurrency and data integrity, persistence of data, etc. The capabilities of this approach can be extended by adding new interfaces to the ServiceModel Packages which are specific to testing i.e. they are only used to support test activities. These interfaces must have a set of operations that enable the internal state of the Service to be externally exposed.

7 Conclusion

The work described in this paper has shown that it is possible to computer generate a set of conformance tests for an interoperability specification from the specification itself using a workflow based upon the usage of Unified Modelling Language (UML). The UML computer-based representation format of the interoperability specification is transformed into an intermediate form that is then used to as the basis for the test generation algorithms. The key findings from this investigation into the capabilities of the auto-generation of service interoperability conformance tests from the specification of the service using the UML are:

- The UML-Profile for service-oriented architectures must be defined and used with the appropriate UML authoring tool for the creation of the service specification. Limitations on the interoperability of file formats between UML authoring tools makes it essential that a tool-independent format tailored to the service UML-Profile, is used;
- The UML-based description of the service must consist of the service model(s), the data model(s), protocol state machine definitions for each of the services and

the interfaces to these services (these must include detailed pre/post-condition statements for each of the transitions) and the set of conformance statements;

- The pre/post-condition statements are used to define the start and end conditions that must be true before and after the system state transition has occurred. Valid precondition and postcondition statements in the protocol state machine(s) are only possible if extra classes are added to the service specification to define the constrained system data structures that are used to describe parameters used in the service operations i.e. these classes have no relevance on the actual data exchanged by the service;
- The testing of an interoperability specification using only the service interface means that the internal state of the end-systems cannot be known, only implied. The only way to observe the internal state of the end-systems is to define extra test interfaces. These test interfaces can then be used as part of the testing approach;
- Testing of a specific implementation requires that the corresponding conformance statement is produced for the service and its component interfaces. An implementation is not necessarily required to support all of the functionality defined in a specification. The test auto-generation tools must then amend the test generation algorithms in accordance with the conformance statement.

Acknowledgements

The author thanks the European Commission for partially funding the work presented in this paper. This funding was through the EU 6th Framework Programme Information Society Technologies (IST) Project No: 507128 – "Technology Enhanced Learning: Conformance – European Requirements & Testing (Telcert)". Further information on Telcert is available at: http://www.opengroup.org/telcert/

References

[1] AA.VV. Scorm 2004: The scorm overview. Technical report, ADL, January 2004. http://www.adlnet.gov/index.cfm.

[2] AA.VV. The open brand - register of certified products. Technical report, The Open Group, 2005.

[3] A.Bertolino. Final recommendations on advanced testing technologies, v1.0. Technical report, Telcert Public Deliverable No. D14, 2006.

[4] J.Warmer A.Kleppe and W.Bast. *MDA Explained - The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.

[5] J.Simon C.Schroeder and C.Smythe. Ims general web services addressing profile v1.0 final release. Technical report, IMS Global Learning Consortium, January 2006. http://www.imsglobal.org/gws/index.html.

[6] J.Simon C.Schroeder and C.Smythe. Ims general web services attachments profile v1.0 final release. Technical report, IMS Global Learning Consortium, January 2006. http://www.imsglobal.org/gws/index.html.

[7] J.Simon C.Schroeder and C.Smythe. Ims general web services base profiles v1.0 final release. Technical report, IMS Global Learning Consortium, January 2006. http://www.imsglobal.org/gws/index.html.

[8] J.Simon C.Schroeder and C.Smythe. Ims general web services security profile v1.0 final release. Technical report, IMS Global Learning Consortium, January 2006. http://www.imsglobal.org/gws/index.html.

[9] J.Simon C.Schroeder and C.Smythe. Ims general web services wsdl binding guidelines v1.0 final release. Technical report, IMS Global Learning Consortium, January 2006. http://www.imsglobal.org/gws/index.html.

[10] C.Smythe. Ims abstract framework: White paper v1.0. Technical report, IMS Global Learning Consortium, July 2003. http://www.imsglobal.org/af/index.html.

[11] C.Smythe. Content repository access service specification & uml profile, v1.0. Technical report, Telcert Public Deliverable No. D15, December 2004. http://www.opengroup.org/telcert/downloads.htm.

[12] D.Carlson. *Modelling XML Applications with UML: Practical e-Business Applications*. Addison-Wesley, 2003.

[13] K.Ballinger et al. Web services interoperability basic profile version 1.1. Technical report, Web Services-Interoperability Organization, August 2004. http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile.

[14] I.Jacobson J.Rambaugh and G.Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 2005.

[15] M.Nottingham. Web services interoperability simple soap binding profile version 1.0. Technical report, Web Services-Interoperability Organization, August 2004. http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile.

[16] S.Wilson and K.Riley. Ims application profile guidelines white paper: Part 1 management overview. Technical report, IMS Global Learning Consortium, October 2005. http://www.imsglobal.org/ap/index.html.

[17] S.Wilson and K.Riley. Ims application profile guidelines white paper: Part 2 technical manual. Technical report, IMS Global Learning Consortium, October 2005. http://www.imsglobal.org/ap/index.html.

Session 3: Composing

Paper 7
Detection of Web Service Substitutability and Composability
Michael D. Ernst, Jeff H. Perkins
(*Massachusetts Institute of Technology - USA*)
Raimondas Lencevicius
(*Nokia - USA*)

Paper 8
Towards Efficient Matching of Semantic Web Service Capabilities
Sonia Ben Mokhtar, Anupam Kaul,
Nikolaos Georgantas, Valerie Issarny
(*INRIA Rocquencourt - France*)

Detection of Web Service Substitutability and Composability

Michael D. Ernst [1]

CSAIL
MIT
Cambridge, MA, USA

Raimondas Lencevicius [2]

Nokia Research Center Cambridge
Cambridge, MA, USA

Jeff H. Perkins [3]

CSAIL
MIT
Cambridge, MA, USA

Abstract

Web services are used in software applications as a standard and convenient way of accessing remote applications over the Internet. Web services can be thought of as remote procedure calls. This paper proposes an approach to determine web service substitutability and composability. Because web services may be unreliable, finding other services substitutable for an existing one can increase application uptime. Finding composable services enables applications to be programmed by composing several web services (using one service's output as an input to another web service). We have implemented our technique and evaluated it on 14 freely available web services producing 92 outputs. Our approach correctly detects all composable and substitutable web services from this set.

Key words: web services, composition, testing, dynamic analysis

[1] Email: mernst@csail.mit.edu
[2] Email: Raimondas.Lencevicius@nokia.com
[3] Email: jhp@csail.mit.edu

1 Web service integration problem

Web services [13] are software building blocks that can be accessed over the Internet in a standard programmatic way using SOAP messaging. They allow programmers to access data from remote providers without extracting it from HTML web pages or using proprietary protocols. Web services are used in various areas: customized stock tracking and trading applications, product search and ordering, address validation, and so on. Programmers can select from over 100 services listed on XMethods.org [1] and other web service providers. Businesses are also adopting and publishing web services for business-to-business communication.

It can be difficult to integrate web services, since most of them were never designed to work together. In theory, semantic information in WSDL [14] files was supposed to solve this problem. In practice, it is often insufficient. In most web services we considered, the web service operation parameter types are indicated simply as strings, floats, and integers. It is impossible to decide from a WSDL file if the input string is a stock ticker or a town name, or what the units of the output quantity are. Some services are even worse, returning a single untyped XML object instead of a typed set of outputs. Because currently WSDL files don't carry enough information to decide substitutability or composability, there is a need for automatic techniques to deduce this information. The web services cover a wide variety of domains. Considering semantic hints from web service names and operations, humans might guess that some operations may be substitutable or composable. Some of those guesses could be wrong. For example, some of the stock quote services return a variety of different results such as the previous closing price, the opening price, the current price, the daily high and low price, the annual high and low price, etc. The only semantic information available is the name of each parameter. Those names are not always clear. One stock service we considered has output parameters named "HIGH" and "LOW". It is not clear whether those are the daily high and low or the annual high and low. Our tool discussed in this paper, however, was able to determine that they were substitutable with output parameters in a different service named "DayHighPrice" and "DayLowPrice".

Our goal is to enable creation of applications that deliver new functionality by integrating web services. In the long run, the integration can indicate which services are compatible with one another; substitute one service for another; and transform inputs or outputs in order to make them compatible. We wish to support the integration performed by programmers writing software, by users who compose services, or even by applications as they discover new services. We discuss each of these scenarios in turn.

Information about web service composability or substitutability can be very useful for programmers. Suppose that a new information source or sink becomes available. As noted above, existing documentation is not necessarily adequate for the programmer's purpose. However, given the information source or sink, and an example application that uses it, tools based on our techniques will enable the programmer to explore the semantics of the feed in order to more quickly build

applications that properly use it.

End users can discover web services and may wish to compose them. Suppose that a user discovers two services created without knowledge of one another, and they do not adhere to a common standard. Tools should enable the user to create a new application on the fly by connecting them. A composition wizard would permit the user to make sensible connections between them, rejecting nonsensical ones, and converting the representation of those with compatible semantics but incompatible formats. For example, a motion detector's output might not be a sensible input to a shopping application, but could be provided as spatial control for aiming a video camera. As another example, today a difference in data format renders an application completely unable to use a data source. Based on observations of use, a future system could infer transformations that retain the meaning. Trivial examples are centigrade–Fahrenheit and polar–Cartesian coordinates, but future work should address more ambitious ones as well.

Currently, the stability of web services is not guaranteed. Finding substitutable web services allows application developers to increase their application uptime. For example, suppose that a blogger posts a local weather report from a home meteorological station. A weather application could notice this new information source and determine that it is (imperfectly) correlated with other weather data, perhaps after transformations. If the primary weather service becomes unavailable, the application automatically converts the blogger's information into a form compatible with the application and uses it to approximate the missing information. As another example, the system could determine when multiple services provide interchangeable functionality and choose the one that is cheapest, fastest, or most accurate, based on user preferences. Such substitutability improves system reliability.

Our work does not solve all of the above problems. However, it takes a step toward their solution by proposing and evaluating techniques for automatically computing web service substitutability and composability.

The paper is organized as follows. Section 2 presents an example of how substitutability and composability can be detected. Section 3 proposes an approach for detecting web service substitutability and composability. Section 4 discusses the web services used in our experiments, and Section 5 shows the experimental results. The paper concludes with related work, future work, and conclusions.

2 Substitutability and composability detection example

Here we present an example of the application of our technique.

Consider two web services Stock1 and Stock2, and suppose they have the partial input/output behavior shown in the tables below, where the leftmost column is the input and the other columns are the output. Those values could have been obtained by testing, by random invocations, by observing their actual user over time, or in any other way.

Service Stock1:

StockTicker	Price
ADBE	36.90
INTC	19.88
MSFT	27.40
QCOM	50.86

Service Stock2:

Stock	LatestPrice	Volume
ADBE	37.00	1.9M
MSFT	27.39	34.9M
QCOM	50.92	30.9M
YHOO	30.80	24.8M

Our technique works in two phases. The first phase detects the overlap of the StockTicker and Stock inputs (ADBE, MSFT, and QCOM are the same inputs) as well as overlap with a margin of error of the Price and LatestPrice outputs. The Volume output of the service Stock2 does not overlap with any other column of inputs or outputs. Since there are no inputs of one service that overlap outputs of another service, there is no potential for composability. However, since there are inputs that overlap inputs and outputs that overlap outputs, the services are potentially substitutable.

The second phase aligns the invocations of the two services for the overlapping inputs and outputs, and finds the best match of different overlapping outputs. In our example, the ADBE 36.9 invocation of Stock1 aligns with the ADBE 37 invocation of Stock2, MSFT 27.40 aligns with MSFT 27.39, and so on. If thus aligned invocations match well (as they do in our example), the services are considered substitutable.

3 Approach

This section describes our algorithm for computing web service composability and substitutability. Our approach assumes that trace data is available from executions of a set of web services. Then it finds similarities between the inputs and outputs of different web services.

A single web service contains one or more operations. Each operation takes zero or more inputs and produces one or more outputs. Our algorithm treats each operation independently. We use the term "param" to mean an input or output.

Our algorithm is a dynamic one; that is, it infers composability and substitutability by observing actual executions of the web service. Simply put, it searches for outputs that match inputs to determine composability, and it searches for inputs that match inputs and outputs that match outputs to determine substitutability. The notion of "matching" is parameterizable; different matching sub-algorithms can be plugged into our framework.

Suppose that there are two operations op_1 and op_2, in different web services. The algorithm observes, at run time, many invocations of op_1 and many invocations of op_2; for each invocation, the observed trace data indicates each param value (input and output value). There are two challenges. First, the algorithm must determine which invocations of op_1 are related to which invocations of op_2. Second, the algorithm must determine which params of op_1 are related to which params of op_2.

As an example, suppose op_1 takes a movie as input, and it outputs the names

and zip codes of theaters showing the movie. Suppose that op_2 takes a zip code and a location as input, and it outputs the approximate distance between them. The first challenge is to determine which invocations of op_1 are related to invocations of op_2. An invocation of op_1 matches an invocation of op_2 if op_1 outputs the same zip code as the one used as input by op_2. Many invocations of op_1 will not match any invocation of op_2, and vice versa. Once the invocations have been aligned, the second challenge is to determine which params are related. In the example, only the zip codes are related, and no other params are.

The algorithm works in two phases. The first phase aligns invocations of different services. The second phase builds on those results and performs a second, potentially looser matching operation to match params. Approximately, the first phase indicates composability (though composability could be refined by the second phase), and the second phase computes substitutability.

3.1 Algorithm

Figure 1 contains the algorithm in pseudocode. The $setfraction$ and $listfraction$ constants are selected empirically.

Both phase 1 and phase 2 of the algorithm perform a value matching step to determine the percentage of matching values. These two invocations of VALUE-MATCH can and should be different. In phase 1 SET-MATCH should use a relatively low $setfraction$, because arbitrary executions are not likely to line up very often, but a rather strict VALUE-MATCH (such as exact equality) to avoid false positives. By contrast, in phase 2 LIST-MATCH should have a high $listfraction$ cutoff (if the invocation matching occurred properly, then any true match should be overwhelmingly common), but the value matching might be made less rigorous (to permit floating-point roundoff rather than requiring an exact match, or to permit different printed representations) to avoid false negatives.

The algorithm treats operations as logical units; for example, if many invocation alignments are possible, the best one is chosen, and if many param matchings are possible, again the best one is chosen. An alternative approach that considers each output as an independent operation would simplify the algorithm but degrade its quality. For example, consider two stock services. Both take a stock ticker and return a number of output parameters such as OpenPrice, DayHighPrice, DayLowPrice, LastPrice, etc. Depending on the volatility of the stock and the time of day the services were executed, many of the output parameters might have very similar values. For example, on a day when prices are rising DayHighPrice may often be very similar to the LastPrice. Thus, there may be many different combinations of output parameters that exceed $listfraction$. It is important to choose the best among them, and not to match any one param to multiple params in another operation.

When at least one input and output param match up, then unmatched params may indicate constant parameters or mappings. As an example of a constant parameter, a movie service that returns the movies playing near a zip code may take a zip code and a radius. The radius is not a critical parameter. It may always have the

▷ **PHASE 1:**
1 **for** every param p_1 from an operation op_1 of a web service s_1
2 **do for** every param p_2 from an operation op_2 of a web service $s_2 \neq s_1$
3 **do if** SET-MATCH$(valuesof(p_1), valuesof(p_2)) > setfraction$
4 **then if** IS-INPUT-AND-OUTPUT(p_1, p_2)
5 **then** mark p_1 and p_2 as composable
6 **else** mark op_1 and op_2 as potentially substitutable
▷ **PHASE 2:**
7 **for** every pair $\langle op_1, op_2 \rangle$ of potentially substitutable operations (from Phase 1)
8 **do** ▷ First, find an alignment between invocations
9 Choose p_1 from op_1 and p_2 from op_2 be such that
IS-INPUT-AND-OUTPUT(p_1, p_2) and SET-MATCH(p_1, p_2) is maximal
10 An invocation of op_1 corresponds to (is aligned with) an invocation of op_2
if VALUE-MATCH(p_1, p_2)
(In the remainder of the algorithm, ignore non-aligned invocations.)
11 ▷ Second, find a mapping among params
12 Choose m to be the mapping between the params of op_1 and op_2
that maximizes $\sum_{p_1 \in op_1}$ LIST-MATCH$(valuesof(p_1), valuesof(m(p_1)))$
13 ▷ Third, mark well-matched params as substitutable
14 **for** each pair $\langle p_1, p_2 \rangle \in m$ ▷ p_1 and p_2 are corresponding params
15 **do if** LIST-MATCH$(p_1, p_2) > listfraction$
16 **then** mark p_1 and p_2 as substitutable.

SET-MATCH(set_1, set_2)

▷ Return the fraction of elements of set_1 and set_2 that match
▷ Example: SET-MATCH$(\{1,2,3\},\{1,3,4,5\}) \rightarrow \min(.67, .5) \rightarrow .5$
$match_1 \leftarrow \{v_1 \in set_1 : \exists v_2 \in set_2 :$ VALUE-MATCH$(v_1, v_2)\}$
$match_2 \leftarrow \{v_2 \in set_2 : \exists v_1 \in set_1 :$ VALUE-MATCH$(v_1, v_2)\}$
return $\min(|match_1| / |set_1|, |match_2| / |set_2|)$

LIST-MATCH$(list_1, list_2)$

▷ Return the fraction of corresponding list elements that match
▷ Example: LIST-MATCH$([1,2,3,4,5],[1,4,3,2,5]) \rightarrow .6$
return $|\{i :$ VALUE-MATCH$(list_1[i], list_2[i])\}| / |list_1|$

VALUE-MATCH(v_1, v_2)

▷ Return true if the two values match
▷ VALUE-MATCH is set by the specific instantiation of our framework. Examples are:
▷ **return** $v_1 = v_2$
▷ **return** $(v_1/v_2 < 1 + \epsilon)$ **and** $(v_2/v_1 < 1 + \epsilon)$
▷ **return** (v_1 is a prefix of v_2) **or** (v_2 is a prefix of v_1)

IS-INPUT-AND-OUTPUT(p_1, p_2)

return (p_1 is an input and p_2 is an output) **or** (p_1 is an output and p_2 is an input)

Fig. 1. Algorithm for determining Web Service composability and substitutability.

same value (as in our experiments), or its value may be more reasonably supplied by the user rather than extracted from the outputs of the service that yielded the zip code. As an example of a mapping, consider two currency exchange services. One takes two country names as input and returns the exchange rate. The other takes two currency names as input and returns the exchange rate. The exchange rate output parameter can be used to line up the results in the first phase of the algorithm, but the input parameters will not match in the second phase. However, a consistent mapping can be found from country name to currency name and vice versa (e.g., "United States" and "USD", "Europe" and "Euro", etc.). Once the mapping is determined, the two services become substitutable.

Duplicated values, which occur frequently in a parameters valueset, carry little information even though they may match well. For example, suppose that two Boolean operations each return $true$ half of the time. These match well, but the mapping carries little information content in terms of matching invocations to one another. So on line 12 of Phase 2, the algorithm discards matches where a single item matches multiple items. An alternative formulation would consider multiple params as necessary, until the matching was unique.

4 Experimental methodology

This section describes the web services and test data used in our experiments.

4.1 Web services

We used 14 different web services and invoked 16 different operations (methods) on them. These operations produce 92 different outputs. A web service operation may produce multiple outputs.

The 92 outputs include 16 outputs that are constants and 6 outputs that are duplicates of an input in the same operation. Our tool ignores constant and duplicate outputs. 10 of the output constants are from the gold operation — this service does not have an input, so it returns the same values every time it is called.

Figure 2 is a synopsis of the web services, their operations, inputs and outputs. Figure 3 gives the WSDL addresses of these web services.

4.2 Test data

We obtained experimental data from each service by calling it 50 times, choosing input parameters at random from a predefined set of possible choices (see Figure 4). We used the same set for each input parameter of the same type. Each combination of input values is used at most once for each service. The number of choices is constrained to generate data similar to the real-world data that would be generated by a few users of a service. For example, users probably would use the service to check for movies in their local geographical area.

Figure 4 shows the values used for each parameter. We use $setfraction$ of

Service & operation	#Inputs & description		#Outputs & descrip.	
stock_wsx.GetQuote	1	ticker	16	quote info
stock_gama.GetLatestStockDailyValue	2	ticker, exchange	1	quote
stock_xmethods.getQuote	1	ticker	1	quote
stock_sm.GetStockQuotes	1	ticker	10	quote
weather_global.GetWeather	2	city, country	10	weather info
currency_exchange.getRate	2	country, country	2	exchange rate
currency_convert.ConversionRate	2	currency, currency	1	exchange rate
gold.GetLondonGoldAndSilverFix	0		10	gold, silver info
region_ab.abbrevToRegion	1	state abbrev	4	state name
region_name.regionToAbbrev	1	state name	4	state abbrev
geoip.GetGeoIP	1	IP address	5	country
location.getCity	1	zip code	1	city
Zip_ripe.ZipCodeToCityState	1	zip code	1	city, state
Zip_ripe_city.CityStateToZipCode	2	city, state	1	zip code
airport.getAirportInfoByAirportCode	1	airport code	16	airport info
movies.GetTheatersAndMovies	1	zip code, radius	6	movie info

Fig. 2. Web services used in our experiments.

Service/operation and WSDL file
stock_wsx.GetQuote http://www.webservicex.com/stockquote.asmx?WSDL
stock_gama.GetLatestStockDailyValue http://www.gama-system.com/webservices/stockquotes.asmx?wsdl
stock_xmethods.getQuote http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl
stock_sm.GetStockQuotes http://www.swanandmokashi.com/HomePage/WebServices/StockQuotes.asmx?WSDL
weather_global.GetWeather http://www.webservicex.com/globalweather.asmx?WSDL
currency_exchange.getRate http://www.xmethods.net/sd/CurrencyExchangeService.wsdl
currency_convert.ConversionRate http://www.webservicex.com/CurrencyConvertor.asmx?wsdl
gold.GetLondonGoldAndSilverFix http://www.webservicex.net/LondonGoldFix.asmx?WSDL
region_ab.abbrevToRegion http://www.synapticdigital.com/webservice/public/regions.asmx?WSDL
region_name.regionToAbbrev http://www.synapticdigital.com/webservice/public/regions.asmx?WSDL
geoip.GetGeoIP http://www.webservicex.com/geoipservice.asmx?WSDL
location.getCity http://webservices.imacination.com/distance/Distance.jws?wsdl
zip_ripe.ZipCodeToCityState http://www.ripedev.com/webservices/ZipCode.asmx?WSDL
zip_ripe_city.CityStateToZipCode http://www.ripedev.com/webservices/ZipCode.asmx?WSDL
airport.getAirportInfoByAirportCode http://www.webservicex.com/airport.asmx?wsdl
movies.GetTheatersAndMovies http://www.ignyte.com/webservices/ignyte.whatsshowing.webservice/moviefunctions.asmx?wsdl

Fig. 3. WSDL files for the web services of Figure 2.

60% and *listfraction* of 80%. Both SET-MATCH and LIST-MATCH use an almost exact VALUE-MATCH function that allows only 1% difference for floating number matches.

stock_wsx.getQuote.input.parameters.symbol stock_gama.GetLatestStockDailyValue.input.parameters.strStock stock_xmethods.getQuote.input.symbol stock_sm.getStockQuotes.input.parameters.QuoteTicker	71 Nasdaq stocks, most of which are in the Standard and Poors 500
stock_gama.GetLatestStockDailyValue.input.parameters.strExchange	constant = "nasdaq"
weather_global.getWeather.input.parameters.CityName	68 Massachusetts airport names
weather_global.getWeather.input.parameters.CountryName	constant = "United States"
currency_exchange.getRate.input.country1 currency_exchange.getRate.input.country2	12 countries
currency_convert.ConverstionRate.input.parameters.FromCurrency currency_convert.ConverstionRate.input.parameters.ToCurrency	151 currencies
region_ab.abbrevToRegion.input.parameters.regionCode	50 US state abbreviations
region_name.regionToAbbrev.input.parameters.regionName	50 US state names
geoip.getGeoIP.input.parameters.IPAddress	all possible IP addresses
location.getCity.input.zip zip_ripe.ZipCodeToCityState.input.parameters.ZipCode movies.GetTheatersAndMovies.input.parameters.zipCode	72 Massachusetts zip codes
Zip_ripe_city.CityStateToZipCode.input.parameters.City	72 Massachusetts cities
Zip_ripe_city.CityStateToZipCode.input.parameters.State	constant = "ma"
airport.getAirportInfoByAiportCode.input.parameters.airportCode	68 airport codes (primarily in MA)

Fig. 4. Values used for testing. Unless otherwise noted, the values for a parameter were chosen at random from the distribution listed in the right column of the table.

5 Experimental results

We have applied our tool to detect and determine substitutability and composability of the web services of Section 4.

Before looking at the results generated by our tool, we first explain what substitutions and compositions can be found in the ideal case. We determined these by careful hand examination of the services, including additional experimentation where necessary.

5.1 Substitutability results

There are 13 possible direct substitutions. All of these are between the various stock services. The two zip code to city conversion services do not substitute because one returns just the city name while the other returns the city and state. The one that returns just the city has a separate operation that returns the state. So in theory it might be possible to substitute one service for the other. However, this is not handled by our tool at the moment. The two currency conversions services do not substitute because one uses countries as input and the other uses currencies as input.

The execution of our tool on the data automatically finds all direct substitutions. The tool does not find any false positives, and there are no false negatives.

5.2 Composability results

There are 6 possible direct compositions. A composition consists of a service whose output is a valid input to a different service. We assume that the output service must provide all of the non-constant parameters to the input service, although in some cases, it might make sense to propose compositions to input parameters independently and allow the user to specify the other parameter values or use separate

services for other parameters.

Two of the compositions have input operations with two parameters. In both cases one of the parameters is a constant in our data, so there is effectively one input for our purposes. Our tests always set the second parameter of the Zip_ripe_city.CityStateToZipCode operation to "Massachusetts". Our tests always set the radius parameter (distance from the zip code of interest) of the movies.GetTheatersAndMovies operation to 0.

The tool finds all direct compositions and does not find any false positives. There are two more complex possible compositions that are not found. currency_exchange.getRate takes two countries, neither of which is constant. geoip.GetGeoIP returns the country for a specific IP address. One could imagine an interesting composition which takes the country from an IP address and calculates the currency exchange with a constant country (e.g., the United States). Our tool does not find this composition because it chooses IP addresses at random and there was not a good correlation with the country names used.

The Zip_ripe.ZipCodeToCityState operation outputs a city/state as a single string such as "Cambridge, MA". The Zip_ripe_city.CityStateToZipCode operation takes two input parameters (city and state). By parsing the city and state from the output of the first operation, the second operation could be composed with it. Our tool does not find compositions that require a single output to be parsed into multiple outputs.

6 Related work

None of the related work discussed in this section uses analysis of runtime information as our approach does.

Dong et al. [2] have built a web service search engine, Woogle, that supports searching for web service operations similar to a given one. The tool also supports searching for web service operations composable with a given one. The tool only uses information available in WSDL files, but clusters it, based on the names of the fields, in an effort to extract semantically meaningful concepts. The work of Dong et al. is orthogonal to ours and could be used as a complement to our dynamic substitutability and composability detection.

Majithia and others [7] propose the Triana toolkit, which allows interactive web service composition. Triana checks the types of parameters in WSDL and even performs type conversions. However, the toolkit does not offer any automatic detection of composability.

Most of the research on discovering web service composability assumes that web services are annotated with semantic information (beyond WSDL) and uses that information to detect substitutability or composability. Such semantic information might be available in the future; however, the web services available now lack it. Sirin, Hendler, and Parsia [11, 12] assume annotations in OWL-S (DAML-S in the first paper). Lassila and Dixit [6] propose a similar scheme using a subset of OWL-S (called DAML-S Lite at the time).

Much research is dedicated to matching user requests to a web service or their composition. This is related to our search for substitutability. Paolucci et al. [9] propose to achieve this with a matching engine using DAML-S service descriptions. Rao, Kungas, and Matskin [10] use a propositional linear logic prover to compose web services according to user requests. Pistore et al. [8] propose a tool WS-Gen; given a set of web services with semantic descriptions and a user request, it generates a composed web service. Kim and Gil [5] propose a tool that interactively guides the user from their request to a composition of web services achieving that request. Their tool uses semantic information to find services composable to the ones already in the composition. Pistore et al. and Kim and Gil's work could be potentially used on the services we find to be composable.

BPEL4WS [4] is a language to specify web service composition. As such it does not address the issue of finding composable services, but is a good tool to implement and present the composed services.

7 Future work

Our promising preliminary results suggest that automatic detection of web service composability and substitutability is a promising direction for future research. However, additional work is required to make the technique practical. Here we note some directions that we plan to pursue.

We would like to experiment with additional web services, including commercial ones. We would also like to apply our techniques to Internet information services that are not packaged as web services, for instance by "screen scraping" the results of web forms. We expect that our technique could also be applied to other software services, and we plan to experiment with components of the Nokia mobile phone architecture.

Our framework is parameterized by matching algorithms. We plan to experiment with more sophisticated matching algorithms. For instance, when provided with sufficient data, an approximate matching algorithm could determine that "$5" and "5" stand for the same quantity, or that the string "5" and the number 5 are the same. Other machine learning techniques, such as those of Daikon [3], could indicate properties of parameters (for example, zip codes are 5-digit strings that are composed solely of digits). Such an approach could also assist in determining when substitutability is not symmetric. For example, a stock service that supports all exchanges can be substituted for one that only handles NASDAQ stocks, but not vice versa; this can be though of as a form of subtyping.

Once our algorithm has aligned invocations and matched parameters, correlations could be inferred among un-matched parameters. For example, if all other parameters match, it could be inferred that "US" in one web service must mean the same thing as "United States" in another. This is just one way to deal with constants and with multiple inputs; we plan other approaches to those problems as well.

We would like to apply our technique to real data collected from the field; our current approach relies on inputs that we made up. Real data will reveal how much

repetition of values occurs and will aid us in tuning our algorithms.

8 Conclusion

With web services becoming standard software building blocks accessible over the Internet, it becomes important to automatically find substitutable and composable services. Finding substitutable web services allows application developers to increase their application uptime by replacing unreliable services on the fly. Finding composable web services helps programmers and users to build interesting applications using web service compositions. This paper describes a method to discover substitutability and composability of web services. We have applied this method to 14 freely available web services. The technique discovered that parameters in 6 pairs of operations are substitutable (for each pair, all non-constant inputs match, and at least one output parameter matches), and 6 additional pairs of services are composable (an output of one service is sensible as an input to the other). Our tool is precise: it does not find any false positives. We hope that our approach will enable more powerful tools for web service programming and use.

Acknowledgments

The authors thank Alexander Ran, Karel Driesen, and the anonymous reviewers for comments on the paper.

References

[1] *Xmethods.org*, 2006.

[2] Xin Dong, Alon Y. Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Simlarity search for web services. In *e)Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB2004)*, pages 372–383, Toronto, Canada, August 31 - September 3 2004.

[3] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Trans. Software Eng.*, 27(2):99–123, 2001.

[4] IBM. *Business Process Execution Language for Web Services version 1.1*.

[5] Jihie Kim and Yolanda Gil. Towards interactive composition of semantic web services. In *in Proceedings of the AAAI Spring Symposium on Semantic Web Services*, 22nd - 24th March 2004.

[6] Ora Lassila and Sapna Dixit. Interleaving discovery and composition for simple workflows. In *in Proceedings of the AAAI Spring Symposium on Semantic Web Services*, 22nd - 24th March 2004.

[7] Shalil Majithia, Matthew S. Shields, Ian J. Taylor, and Ian Wang. Triana: A graphical web service composition and execution toolkit. In *in Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 514–523, San Diego, California, June 6-9 2004.

[8] M.Pistore, P.Bertoli, E.Cusenza, A.Marconi, and P.Traverso. Ws-gen: A tool for the automated composition of semantic web services, November 7-11 2004.

[9] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In *The Semantic Web - ISWC 2002, First International Semantic Web Conference*, pages 333–347, Sardinia, Italy, June 9-12 2002.

[10] Jinghai Rao, Peep Küngas, and Mihhail Matskin. Logic-based web services composition: From service description to process model. In *in Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 446–453, San Diego, California, June 6-9 2004.

[11] Evren Sirin, James Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In *In Proceedings of Web Services: Modeling, Architecture and Infrastructure Workshop at ICEIS 2003*, Angers, France, April 2003.

[12] Evren Sirin, Bijan Parsia, and James A. Hendler. Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4):42–49, 2004.

[13] World Wide Web Consortium (W3C). *Web Services Activity*, 2006.

[14] World Wide Web Consortium (W3C). *Web Services Description Language (WSDL)*, 2006.

Towards Efficient Matching of Semantic Web Service Capabilities

Sonia Ben Mokhtar, Anupam Kaul
Nikolaos Georgantas and Valérie Issarny

INRIA Rocquencourt,
Domaine de Voluceau,
BP 105, 78153 Le Chesnay Cedex, France
{Sonia.Ben_Mokhtar,Anupam.Kaul,Nikolaos.Georgantas,Valerie.Issarny}@inria.fr

Abstract

Web services are becoming an incontrovertible paradigm for the development of large scale distributed systems. Combined with semantic Web technologies, in particular ontologies, Web services capabilities can be unambiguously interpreted and automatically used. Nevertheless, efficient matching of semantic Web service capabilities remains an open issue towards the wide acceptance of semantic Web services. In this paper, we analyze the cost of semantic reasoning based on ontologies, which is at the heart of the matching process, and we propose an approach towards efficient matching of semantic Web service capabilities. Our approach introduces optimizations of the matching process at two levels: at the semantic reasoning level in order to reduce the cost of matching concepts within ontologies, and at the matching level, in order to reduce the number of matching iterations over a registry of services.

International Workshop on Web Services Modeling and Testing (WS-MaTe 2006)

1 Introduction

Web services are becoming an incontrovertible paradigm for the development of large scale distributed systems. Indeed, Web services allow a homogeneous use of heterogeneous software components deployed in large networks and in particular the Internet. Using this paradigm, software components are abstracted as Web services; they are described in a declarative manner using the Web Services Description Language (WSDL) and communicate using standard protocols such as the Simple Object Access Protocol (SOAP) on top of Internet protocols (HTTP, SMTP). While the Web services paradigm addresses substantially the heterogeneity issue that arises at the platform layer in distributed applications, another issue remains, which is *syntactic heterogeneity*. Indeed, interaction with Web services is based on the syntactic conformance of required with provided interfaces, for which common understanding is hardly achievable in large scale environments. A promising approach towards addressing syntactic heterogeneity relies on the semantic modeling of service capabilities. This concept underpins the Semantic Web [1]. Semantic modeling for the Web is based on the use of ontologies and ontology languages that support formal description and reasoning on ontologies. A natural evolution has been the combination of the Semantic Web and Web Services into Semantic Web Services. In this area, the Ontology Web Language for Services (OWL-S) [1] is one of the most prominent efforts for describing semantic Web services.

Web services can be advertised in centralized registries (e.g., UDDI), which facilitate Web services discovery and selection in the large network. In these registries, Web services discovery decomposes into two main functions that are: (1) a publishing function, which allows services to be advertised and integrated in the registry; and (2) a querying function, in which functional capabilities required by the user are matched with capabilities provided by the services hosted by the registry. While there already exist various protocols enabling Web services discovery, including UDDI, effectively enabling discovery of semantic Web services remains an open issue, in part due to the challenges posed by semantic reasoning.

The objective of this paper is to analyze the impact of introducing semantic Web technologies in the process of Web services discovery and introduce base principles towards efficient semantic Web services discovery. We further concentrate on registry-based protocols for the convenience of presentation. In the remainder of this paper, we first define in Section 2, the baseline of matching semantic Web services, which is at the heart of semantic Web services discovery and we analyze the impact of on-line reasoning on the performance of service discovery. Then, we present existing efforts towards the optimization of the semantic matching process in Section 3. Building upon these efforts, we propose an approach towards efficient semantic service discovery and selection in Section 4. Finally, we conclude with a summary of our contribution and future work in Section 5.

[1] OWL-S: Semantic Markup for Web Service. http://www.daml.org/services/owl-s

2 Semantic Matching of Service Capabilities

Semantic service matching allows the selection of services providing capabilities that are semantically equivalent to some requested capabilities. As previously identified by Zaremski and Wing in [12], semantically matching capabilities provided by software components decomposes into *signature matching* and *specification matching*. Signature matching deals with the identification of subsumption [2] relationships between the concepts describing inputs and outputs of capabilities. Specification matching deals with matching pre- and post-conditions that describe the functional semantics of capabilities.

2.1 Approaches to semantic matching of service capabilities

A number of research efforts have been conducted in the area of matching semantic Web service capabilities.

A base algorithm for signature matching has in particular been proposed by Paolucci *et al.* in [7]. This algorithm allows matching a requested capability described as a set of provided inputs and required outputs with capabilities, also described as a set of required inputs and provided outputs. Inputs and outputs of capabilities are described with concepts in ontologies. Then, the algorithm defines four levels of matching between two ontology concepts, being respectively a provided and a required concept. These levels are:

- *exact*: if the concepts are the same or if the required concept is a direct subclass of the provided one,
- *plug in* if the provided concept subsumes the required one,
- *subsumes* if the required concept subsumes the provided one, and
- *fail* if there is no subsumption relation between the two concepts.

Then, the matching algorithm scores service descriptions according to the matching levels found between the concepts used in the service request and those used in the service advertisement.

Other solutions to signature matching of semantic Web services have been proposed in the literature [3, 6, 11, 5], these are based on the above.

Specification matching of semantic Web services has been studied in the literature [13, 8, 10, 2]. For instance, in [13], specification matching is performed using theorem proving, i.e., inferring general subsumption relations between logical expressions that specify pre- and post-conditions of services. A more practical way to perform specification matching is to use *query containment* [8, 10, 2]. This is done by modeling both service advertisements and service requests as queries with a set of constraints (e.g., required inputs and outputs are modeled as restrictions on their types). Then, starting from the specified constraints, the possible values of both queries are evaluated, and possible inclusions between the results of the

[2] subsumption: incorporating something under a more general category

queries is inferred. Specifically, a query q_1 is contained in q_2 if all the answers of q_1 are included in the answers of q_2.

Whether semantic matching is performed according to signature or specification matching, or both, the key issue in efficient matching lies in the performance of the underlying semantic reasoning over ontologies, as analyzed below.

2.2 Analyzing the cost of automated semantic matching

To analyze the cost of semantic matching of service capabilities, we consider a UDDI-like registry of Web services described using OWL-S. Specifically, for the sake of simplicity, we provide an evaluation of the signature matching of user requests with the services hosted in the registry. Nevertheless, as specification matching also lies in the semantic reasoning over ontologies, we expect that results obtained for signature matching will also apply to specification matching. Further experiments that include specification matching will be performed in future work. The semantic matching between a requested capability and advertised ones, performed by the repository, is defined by the following matching algorithm.

The requested capability (Req) has a set of provided inputs in_{Req} and a set of expected outputs out_{Req}, whereas each advertised capability (Adv) has a set of expected inputs in_{Adv} and a set of provided outputs out_{Adv}. The matching algorithm uses a numeric function, i.e., $Subsumption(x, y)$, which informs whether the concept x is related to the concept y with respect to the ontology in which they are defined. More precisely, $Subsumption(x, y)$ returns 0 if there is no relationship between x and y, it returns 1 if x subsumes y, and 2 if x and y belong to the same concept, i.e., $exact$ match. The matching algorithm evaluates :

- the predicate $Match(Adv, Req)$, which specifies whether the advertisement Adv matches the request Req, and
- the value $DegreeOfMatch(Adv, Req)$, which specifies to which degree the advertisement matches the request.

This value corresponds to the sum of the results given by the function $Subsumption$ each time a couple of inputs (resp. outputs) are matched. This function allows scoring service advertisements in order to select services with the maximum number of $exact$ matches. The predicate $Match$ and the value $DegreeOfMatch$ are defined as follows:

- $Match(Adv, Req) = (\forall in \in in_{Adv}, \exists in' \in in_{Req} : Subsumption(in', in) > 0)$ and $(\forall out' \in out_{Req}, \exists out \in out_{Adv} : Subsumption(out, out') > 0)$
- $DegreeOfMatch(Adv, Req) = \sum[Subsumption(in_{Req}, in_{Adv}) + Subsumption(out_{Adv}, out_{Req})]$
  for all in_{Req}, out_{Req} in Req and in_{Adv}, out_{Adv} in Adv such that $Subsumption(in_{Req}, in_{Adv}) > 0$ and $Subsumption(out_{Adv}, out_{Req}) > 0$

For a given service request, the above algorithm is performed for all the service advertisements hosted by the registry. The selection is then done using the max-

imum degree of match obtained during the matching process, i.e., we select the service advertisement Adv such that :
$DegreeOfMatch(Adv, Req) = Max(DegreeOfMatch(AdvX, Req))$, for all the advertisements $AdvX$ in the registry.

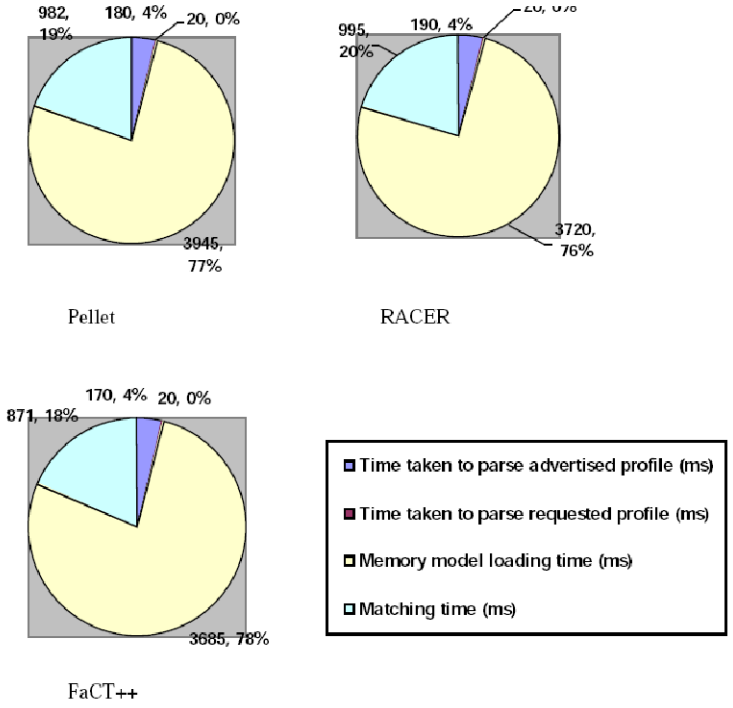


Fig. 1. Time taken to match a request and an advertised service for 7 Inputs, 3 Output

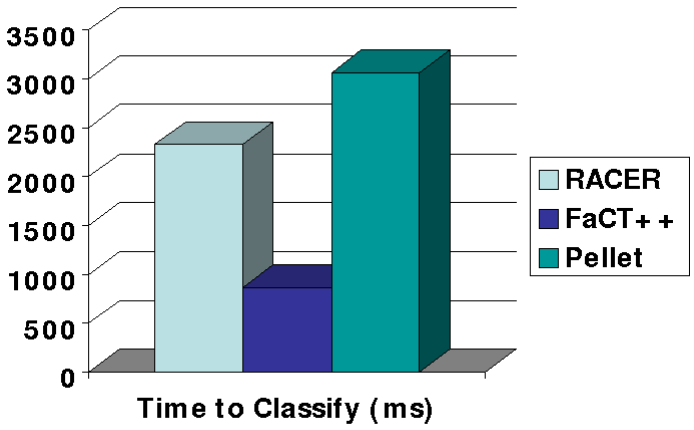


Fig. 2. Time to load and classify an ontology

We have implemented the matching algorithm on a Toshiba Satellite notebook with 1.6 GHz Intel Centrino processor with 512 MB of RAM. Our prototype implementation includes the use of a description logic reasoner (DL-reasoner) to infer

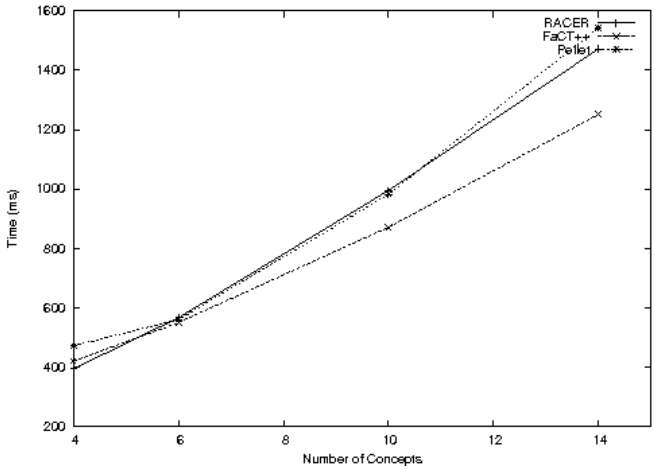


Fig. 3. Time to match concepts using different reasoners

the subsumption relationships between concepts. There are various implementations of DL-reasoners, the most popular ones are : Racer [3], Fact++ [4] and Pellet [5]. We provide a performance evaluation of our prototype implementation using the aforementioned three reasoners in order to assess their impact on the matching tool. We have conducted three main experiments. Figure 1 shows the results of the first experiment. This experiment gives an overview of the cost of each step of the matching process, i.e., (1) time to parse the service advertisement; (2) time to parse the service request; (3) time to load and classify the ontologies involved in the service and request descriptions and (4) time to match the concepts involved in the advertisement and the request, i.e., finding the relationships between concepts within the classified ontologies. In this experiment, the service request is composed of seven provided inputs and three requested outputs. The ontology used for the experiment can be found on-line [6]. This ontology contains 99 OWL Classes, 4 Datatype Properties, 11 Object Properties, 24 Annotation Properties and 5 Individuals. This experiment has been realized using each of the aforementioned reasoners. Results show that the most expensive phase in the process of matching service capabilities is the phase of loading and classifying the involved ontologies (from 76% to 78%).

The second experiment compares the time taken by each of the three aforementioned reasoners to classify the used ontology. Results, depicted in Figure 2, show that the reasoner Fact++ has better results than the other reasoners.

The last experiment that we conducted compares the time taken by each reasoner to match the concepts involved in the service request and the service advertisement within the classified ontology. In this experiment we increased the number of concepts involved in the service request from 4 to 14. Results, depicted in Figure

[3] Racer: http://www.sts.tu-harburg.de/ r.f.moeller/racer/
[4] Fact++: http://owl.man.ac.uk/factplusplus/
[5] Pellet: http://www.mindswap.org/2003/pellet/
[6] http://www.co-ode.org/ontologies/pizza/2005/05/16/pizza.owl

3, show that Fact ++ has better results than the other reasoners.

From the above experiments we can notice that matching semantic service capabilities is a heavy process, which cannot be performed fully on-line. Indeed, for a registry of n services, the time to match a service request with all the services in the registry (in the aim of selecting the service that best fits the request) is equal to Time-to-parse-the-request + n * Time-to-parse-an-advertisement+ Time-to-load-and-classify-ontologies + n * Time-to-match-the-concepts. For n=1, i.e., one service advertisement, and a request composed of 10 concepts, this time is in the order of 4 to 5 seconds using any reasoner. Thus, a number of optimizations have to be introduced towards efficient matching of semantic Web service capabilities.

3 Optimizing Matching of Semantic Web Service Capabilities

As shown in the previous section, on-line reasoning on semantic service descriptions is a costly task. Towards efficient matching of semantic Web services, optimizations can be introduced at two levels:(1) at the semantic ontology reasoning level and (2) at the matching level, as briefly surveyed below. Optimizations at the semantic reasoning level aim at reducing the time to load and classify ontologies, which is the most costly phase in the discovery process. This can be done by various mechanisms, e.g., anticipate the user requests for pre-fetching and pre-classifying ontologies, encoding classified ontologies like in [2]. Optimizations at the matching level aim at reducing the number of semantic matches performed in the querying phase. Possible approaches include pre-computing matching information at publishing phase like in [9], classifying service advertisements.

Optimization at the semantic reasoning level has been proposed by Constantinescu *et al.* in [2]. The authors emphasize the need of efficient indexes and search structures for directories. Towards this goal, they propose to numerically encode service descriptions given in OWL-S. This is done by numerically encoding ontology class and property hierarchies by intervals. More precisely, each class (resp. property) in a classified hierarchy is associated with an interval. Then, each service description maps to a graphical representation in the form of a set of rectangles defined by the sets of intervals representing properties combined with the set of intervals representing classes. Furthermore, for efficient service retrieval, the authors base their work on techniques for managing multidimensional data being developed in the database community. More precisely, they use the Generalized Search Tree (GiST) algorithm proposed by Hellerstein in [4] for creating and maintaining the directory of numeric services. Combining both encoding and indexing techniques allows performing efficient service search, in the order of milliseconds for trees of 10000 entries. However, performance for insertion within trees of the previous size is still a heavy process that takes approximately 3 seconds.

Optimization at the matching level is introduced by Srinivasan *et al.* in [9], which proposes an approach to optimize service discovery in a UDDI registry augmented with OWL-S for the description of semantic Web services. This approach is based on the fact that the publishing phase is not a time critical task. There-

fore, the authors propose to exploit this phase to pre-compute and store information about the incoming services. More precisely, a taxonomy that represents the subsumption relationships between all the concepts in the ontologies used by services is maintained. Then, each concept C in this taxonomy is annotated with two lists, one to store information about inputs of services while the other one is used to store information about outputs of services. More precisely, for each concept in the taxonomy, these lists specify to what degree any request pointing to that concept would match the advertisement. For example, for a particular concept C in the taxonomy, the list storing information about outputs is represented as $[< Adv_1, exact >, < Adv_2, subsumes >, ...]$, where Adv_i points to a service advertisement in the repository and $exact$ (resp. $subsumes$) specify the degree of match between C and the related concept in the corresponding advertisement. A performance evaluation of this approach shows that the publishing phase using this algorithm takes around seven times the time taken by UDDI to publish a service, under the assumption that no additional ontologies have to be loaded to the registry. On the other hand, the time to process a query is in the order of milliseconds.

While the above increases the time spent for publishing service advertisements, it considerably reduces the time spent to answer a user request compared to approaches based on on-line reasoning (e.g., see Figure 1). Indeed, the querying phase is reduced to perform lookups in the hierarchical data structure that represents the classified ontology, and to perform intersections between the lists that store information about the service advertisements. Thus, no on-line reasoning is required to answer a user request.

While the above approaches introduce optimizations, respectively at semantic reasoning and the matching levels, we believe that a solution that integrates optimizations at both levels would obtain better results.

4 Towards Efficient Matching of Semantic Web Service Capabilities

In this section we describe a solution towards the efficient matching of semantic service capabilities. This approach combines optimizations of the discovery process at both the reasoning and the matching levels. Towards the optimization of the discovery process at the reasoning level, we build upon the aforementioned solution proposed by Constantinescu *et al.* for encoding concept hierarchies. Furthermore, in order to perform efficient service matching, we propose to classify service capabilities into hierarchies of similar capabilities.

4.1 Encoding concept hierarchies

In order to avoid semantic reasoning at runtime we propose to encode classified ontologies, represented by hierarchies of concepts, using intervals as described in [2]. These hierarchies represent the subsumption relationships between all the concepts in the ontologies used in the directory. The main idea of the encoding is

that any concept in a classified ontology is associated with an interval. These intervals can be contained in other intervals but are never overlapping. The intervals are defined using a linear inverse exponential function $linKinvexpP(x) = \frac{1}{p^{int(\frac{x}{k})}} + (x\%k) * \frac{1}{k} * \frac{1}{p^{int(\frac{x}{k})}}$, where p and k are two parameters to be fixed. About the scalability of this encoding solution, experiments show that for p=2 and k=5, and a system encoding real numbers as 64 bits doubles, the maximum number of entries that we can have on the first level of the hierarchy is 1071 and the maximum number of levels that we can have on the first entries of a level is 462 levels. Figure 4 taken from [2], shows an example of encoding a hierarchy of concepts with intervals.

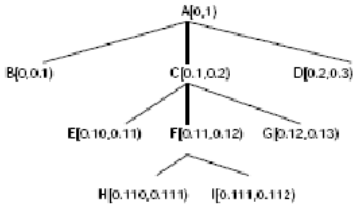


Fig. 4. Example of encoding a class hierarchy

Under the assumption that the classified ontologies are encoded and that service advertisements and service requests already contain the codes corresponding to the concepts that they involve, semantic service reasoning reduces to a numeric comparison of codes. Indeed, to infer whether a concept C1 represented by the interval I1 subsumes another concept C2 represented by the interval I2, it is sufficient to compare whether I1 is included in I2. In order to ensure consistency of codes along with the dynamics and evolution of ontologies, service advertisements and service requests specify the version of the codes being used. We assume that services periodically check the version of codes that they are using and update their codes in the case of ontology evolution.

4.2 Classification of Web service capabilities

Towards the optimization of the number of matches performed to answer a user request we propose to group capabilities provided by networked services into hierarchies of capabilities. Thus, the repository will be structured into groups of similar capabilities. These hierarchies are represented using directed acyclic graphs (DAG). The relationship between capabilities that we consider to construct these graphs is defined by the predicate $Match$ and the value $DegreeOfMatch$ introduced in Section 2.2. Specifically, if $Match(C1, C2)$ and $Match(C2, C1)$ hold and
$DegreeOfMatch(C1, C2)$ = $DegreeOfMatch(C2, C1)$ = 2*(number of outputs of $C1$ + number of inputs of $C2$), i.e., all the inputs of $C2$ are exactly matched with inputs of $C1$ and all the outputs of $C1$ are exactly matched with outputs of $C2$, then $C1$ and $C2$ will be represented by a single vertex in the graph. For all the other cases of

$DegreeOfMatch(C1, C2)$ where $Match(C1, C2)$ holds, $C1$ and $C2$ will be represented in the graph by two distinct vertices with a directed edge from $C1$ to $C2$. Figure 6 shows and example of a DAG representing a classification of capabilities. Note that the function $Match$ is implemented using the encoding techniques discussed above.

The main advantage of using this classification of capabilities is to reduce the number of matches to be processed during the service discovery step. Indeed, if a matching of a requested capability with a capability situated on top of a hierarchy, i.e., a vertex without predecessors, fails, we can infer that it will also fail with all the other capabilities contained in the sub-hierarchy of this capability in the graph, i.e., all the capabilities represented by vertices such that there is a path from the considered vertex to these vertices. On the other hand, if a matching between a requested capability and a capability situated at the bottom of a hierarchy, i.e., a vertex without successors, succeeds, we can infer that the matching will also succeed with all the predecessors of this capability, i.e., all the capabilities represented by vertices from which there is a path from these vertices to the considered vertex. This is expressed by the following two properties :

- (Prop1) : $\neg Match(C, Req) \Rightarrow \forall C'$ such that $Match(C, C') : \neg Match(C', Req)$
- (Prop2) : $Match(C, Req) \Rightarrow \forall C'$ such that $Match(C', C) : Match(C', Req)$

(Prop1) expresses the fact that if a matching of a requested capability Req fails with a capability C, then the matching will also fail with all the capabilities C' that match C, i.e., such that $Match(C, C')$. Using this property along with the transitivity property of the predicate $Match$ we can infer that if a matching with a capability fails, it will fail with all the capabilities of the sub-hierarchy of this capability in the graph.

(Prop2) expresses the fact that if a matching of a requested capability Req successes with a capability C, then the matching will also succeed with all the capabilities C' that are matched by C, i.e., such that $Match(C', C)$. Using this property along with the transitivity property of the predicate $Match$ we can infer that if the matching with a capability successes it will succeed with all the capabilities of the super-hierarchy of this capability in the graph.

The proofs of properties (Prop1) and (Prop2) and the transitivity of the predicate $Match$ are given below.

First we need to proof the transitivity of the function $Subsumption$:
Consider $c1, c2, c3$ three concepts in an ontology :
$Subsumption(c1, c2) > 0$ and $Subsumption(c2, c3) > 0 \Rightarrow Subsumption(c1, c3) > 0$
Assume that $Subsumption(c1, c2) > 0$ and $Subsumption(c2, c3) > 0$. Four cases are possible :

$Subsumption(c1, c2) = 2$ and $Subsumption(c2, c3) = 2$ (1)
$Subsumption(c1, c2) = 2$ and $Subsumption(c2, c3) = 1$ (2)
$Subsumption(c1, c2) = 1$ and $Subsumption(c2, c3) = 2$ (3)

$Subsumption(c1, c2) = 1$ and $Subsumption(c2, c3) = 1$ (4)

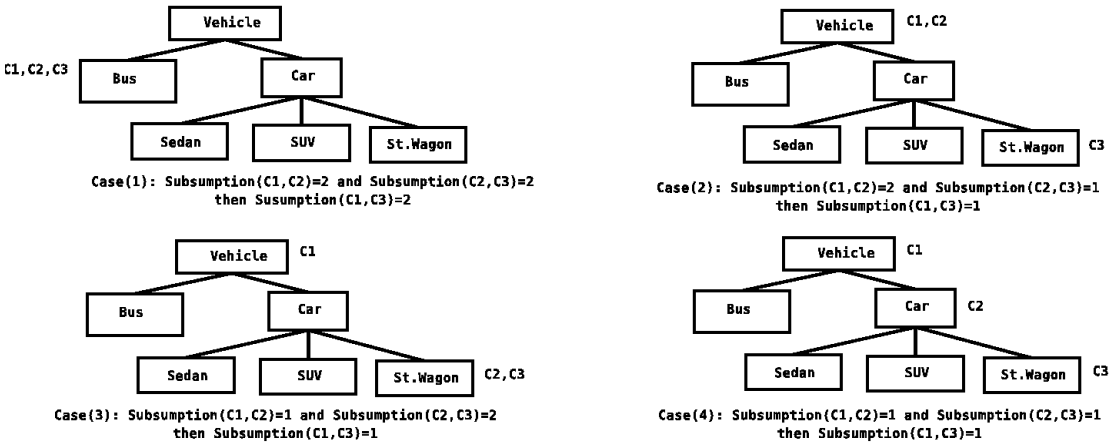


Fig. 5. Transitivity of the function Subsumption

In the case (1), as shown in figure 5, the three concepts refer to the same class in the ontology, then, it is obvious that $Subsumption(c1, c3) = 2$. In both cases (2) and (3), two of the three concepts are equivalent, thus, if one of them subsumes (resp. is subsumed by) the third concept, the second concept will also subsume (resp. be subsumed by) the third one. In these two cases $Subsumption(c1, c3) = 1$. In the last case, one can notice that the second concept is subsumed by the first one, i.e., it is situated in the sub-hierarchy of the first concept within the classified ontology. Then, the third concept is, in its turn, subsumed by the second one, i.e., it is situated in the sub-hierarchy of the second concept. Thus, because of the hierarchical structure of the classified ontology we infer that the third concept is in the sub-hierarchy of the first one, and that $Subsumption(c1, c3) = 1$.

4.2.1 Proof of (Prop1) by contradiction

(Prop1):$\neg Match(C, Req) \Rightarrow \forall C'$ such that $Match(C, C') : \neg Match(C', Req)$
Assume :

$\neg Match(C, Req)$ (1)
and
$\exists C'$ such that $Match(C, C') : Match(C', Req)$ (2)
From (1) and the definition of the matching between capabilities, we can derive that :
$\exists i_C \in In_C, \exists i_{req} \in In_{Req} : Subsumption(i_{req}, i_C) = 0$ (3)
OR
$\exists o_C \in Out_C, \exists o_{req} \in Out_{Req} : Subsumption(o_C, o_{req}) = 0$ (4)
From (2) we can derive that :
$\forall i_{C'} \in In_{C'}, \exists i_{req} \in In_{Req} : Subsumption(i_{req}, i_{C'}) > 0,$ (5)
(because $Match(C', Req)$)
$\forall o_{req} \in Out_{Req}, \exists o_{C'} \in Out_{C'} : Subsumption(o_{C'}, o_{req}) > 0,$ (6)
(because $Match(C', Req)$)

$\forall i_C \in In_C, \exists i_{C'} \in In_{C'} : Subsumption(i_{C'}, i_C) > 0,$ (7)
(because $Match(C, C')$)
$\forall o_{C'} \in Out_{C'}, \exists o_C \in Out_C : Subsumption(o_C, o_{C'}) > 0,$ (8)
(because $Match(C, C')$)
From (5), (7) and the transitivity of the function Subsumption, we can infer that :
$\forall i_C \in In_C, \exists i_{req} \in In_{Req} : Subsumption(i_{req}, i_C) > 0$ (9)
From (6), (8) and the transitivity of the function Subsumption we can infer that :
$\forall o_{req} \in Out_{Req}, \exists o_C \in Out_C : Subsumption(o_C, o_{req}) > 0)$ (10)

However, we know that either (3) or (4) holds. If (3) holds, then, there will be a contradiction with (9). On the other hand, if (4) holds, then, there will be a contradiction with (10). Thus, the assumption (2) is false and the proposition (Prop1) is true.

4.2.2 Proof of (Prop2)

(Prop2): $Match(C, Req) \Rightarrow \forall C'$ such that $Match(C', C) : Match(C', Req)$

Proving (Prop2) can be done by proving the transitivity of the predicate $Match$ that is defined as follows :
Consider $C1, C2, C3$ three capabilities : $Match(C1, C2)$ and $Match(C2, C3) \Rightarrow Match(C1, C3)$
Assume that :

$Match(C1, C2)$ (1)
and
$Match(C2, C3)$ (2)
From (1) we can derive that :
$\forall i_{C1} \in In_{C1}, \exists i_{C2} \in In_{C2} : Subsumption(i_{C2}, i_{C1}) > 0,$ (3)
$\forall o_{C2} \in Out_{C2}, \exists o_{C1} \in Out_{C1} : Subsumption(o_{C1}, o_{C2}) > 0,$ (4)
From (2) we can derive that :
$\forall i_{C2} \in In_{C2}, \exists i_{C3} \in In_{C3} : Subsumption(i_{C3}, i_{C2}) > 0,$ (5)
$\forall o_{C3} \in Out_{C3}, \exists o_{C2} \in Out_{C2} : Subsumption(o_{C2}, o_{C3}) > 0,$ (6)
From (3), (5) and the transitivity of the function Subsumption we can infer that :
$\forall i_{C1} \in In_{C1}, \exists i_{C3} \in In_{C3} : Subsumption(i_{C3}, i_{C1}) > 0$ (7)
From (4), (6) and transitivity of the function Subsumption we can infer that :
$\forall o_{C3} \in Out_{C3}, \exists o_{C1} \in Out_{C1} : Subsumption(o_{C1}, o_{C3}) > 0$ (8)

Finally, (7) and (8) imply that : $Match(C1, C3)$, by definition.

Then, using the transitivity of the predicate $Match(C, Req)$, we infer that for each capability C' where $Match(C', C) : Match(C', Req)$

We use the above properties for both the publishing and the querying functions of the repository as described below.

4.3 Publishing phase

When a new service appears in the network, capabilities provided by this service have to be classified in the repository in order to allow efficient service discovery. The classification of the new capabilities is done as follows: assume $NewC$ is one of the capabilities provided by a new service, and G one of the hierarchy graphs in this repository. First, a semantic matching of $NewC$ has to be performed with each capability of the set Roots(G), i.e., the set of vertices of G that do not have any predecessor in G. If the matching fails with all these capabilities, we can infer that $NewC$ will not have any predecessor in the graph G (Prop1). The second step is to match $NewC$ with each capability of the set Leaves(G), i.e., the set of vertices of G that do not have any successor in G. If the matching fails with all these capabilities, then we can infer that $NewC$, will not have any successor in the graph G. This is specified by the following property proved hereafter:

- (Prop3): $\neg Match(NewC, C) \Rightarrow \forall C'$ such that $Match(C', C) : \neg Match(NewC, C')$

Consequently, to detect if a capability cannot be inserted within a group it is sufficient to match this operation with the Roots and Leaves of the group, rather than matching with all the capabilities of this group. The other cases, when a matching is recognized between the new capability and one of the capabilities of Roots(G) or Leaves(G), are handled according to the algorithm given in Figure 7.

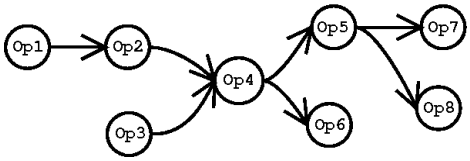
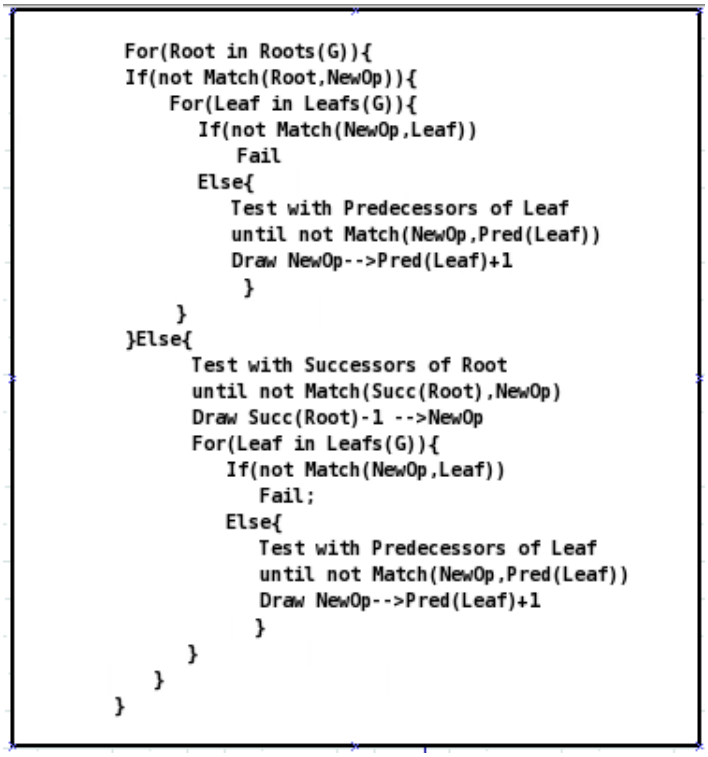


Fig. 6. Operations Grouping Example

4.3.1 Proof of (Prop3) by contradiction

What we want to proof is that :
$\neg Match(NewC, C) \Rightarrow \forall C'$ such that $Match(C', C) : \neg Match(NewC, C')$
Assume that :

$\neg Match(NewC, C)$ (1)
and
$\exists C'$ such that $Match(C', C) : Match(NewC, C')$ (2)
From (1) and the definition of the matching between capabilities, we can derive that :
$\exists i_{nop} \in In_{NewC}, \exists i_C \in In_C : Subsumption(i_C, i_{nop}) = 0$ (3)
OR
$\exists o_C \in Out_C, \exists o_{nop} \in Out_{NewC} : Subsumption(o_{nop}, o_C) = 0$ (4)
From (2), we can derive that :
$\forall i_{nop} \in In_{NewC}, \exists i_{C'} \in In_{C'} : Subsumption(i_{C'}, i_{nop}) > 0,$ (5)
(because $Match(NewC, C')$)
$\forall o_{C'} \in Out_{C'}, \exists o_{nop} \in Out_{NewC} : Subsumption(o_{nop}, o_{C'}) > 0,$ (6)
(because $Match(NewC, C')$)



```
For(Root in Roots(G)){
If(not Match(Root,NewOp)){
    For(Leaf in Leafs(G)){
       If(not Match(NewOp,Leaf))
           Fail
       Else{
          Test with Predecessors of Leaf
          until not Match(NewOp,Pred(Leaf))
          Draw NewOp-->Pred(Leaf)+1
           }
     }
}Else{
      Test with Successors of Root
      until not Match(Succ(Root),NewOp)
      Draw Succ(Root)-1 -->NewOp
      For(Leaf in Leafs(G)){
         If(not Match(NewOp,Leaf))
            Fail;
         Else{
            Test with Predecessors of Leaf
            until not Match(NewOp,Pred(Leaf))
            Draw NewOp-->Pred(Leaf)+1
           }
      }
   }
}
```

Fig. 7. Algorithm of introducing new capabilities

$\forall i_{C'} \in In_{C'}, \exists i_C \in In_C : Subsumption(i_C, i_{C'}) > 0,$ (7)
(because $Match(C', C)$)
$\forall o_C \in Out_C, \exists o_{C'} \in Out_{C'} : Subsumption(o_{C'}, o_C) > 0,$ (8)
(because $Match(C', C)$)
From (5), (7) and the transitivity of the function Subsumption, we can infer that :
$\forall i_{nop} \in In_{NewC}, \exists i_{req} \in In_{Req} : Subsumption(i_C, i_{nop}) > 0$ (9)
From (6), (8) and the transitivity of the function Subsumption we can infer that :
$\forall o_C \in Out_C, \exists o_{nop} \in Out_{NewC} : Subsumption(o_{nop}, o_C) > 0)$ (10)

However, we know that either (3) or (4) holds. If (3) holds, then, there will be a contradiction with (9). On the other hand, if (4) holds, then, there will be a contradiction with (10). Thus, the assumption (2) is false and the proposition (Prop3) is true.

Structuring a registry of services into hierarchies of capabilities allows to reduce the number of semantic matches performed both during the publishing and the querying phases of the service discovery process. Moreover, combined with the numeric encoding of service descriptions, which reduces semantic reasoning to a numeric comparison of codes, efficiency of semantic Web service discovery should be considerably improved.

5 Conclusion and Future Work

Web services allow the rapid development of large scale distributed systems, enabling the interoperation of heterogeneous deployed components. Nevertheless, this introperation is based on the syntactic conformance of Web services interfaces, which somehow restricts the ability to automatically exploit Web service capabilities. Towards the automation of Web service consumption, semantic Web services allows a common understanding of Web service capabilities, which ensures unambiguous service discovery and selection. However, mainly due to the complexity of the underlying semantic reasoning, matching semantic Web service capabilities is a heavy process. In this paper we analyzed the cost of semantic matching of Web service capabilities and we proposed an approach towards efficient service matching. This approach introduces optimizations of the matching process at two levels. First, an optimization at the semantic reasoning level by encoding classified ontologies, reduces the semantic reasoning at runtime to a numeric comparison of codes. Second, optimization at the matching level, by structuring a registry of services into hierarchies of similar services, allows reducing the number of matches to be performed at runtime, to a subset of the registry services. A prototype implementation of our approach including specification matching is currently under development.

References

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[2] Ion Constantinescu and Boi Faltings. Efficient matchmaking and directory services. In *The 2003 IEEE/WIC International Conference on Web Intelligence (WI'03)*, Halifax, Canada, October 2003.

[3] J. G. Pereira Filho and M. van Sinderen. Web service architectures - semantics and context-awareness issues in web services platforms. Technical report, Telematica Instituut, 2003.

[4] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized search trees for database systems. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *Proc. 21st Int. Conf. Very Large Data Bases, VLDB*, pages 562–573. Morgan Kaufmann, 11–15 1995.

[5] Gonzalez-Castillo Javier, Trastour David, and Bartolini Claudio. Description logics for matchmaking of services. In *Proceedings of the of the KI-2001, Workshop on Applications of Description Logics Vienna, Austria*, volume 44, September 2001.

[6] Shalil Majithia, David W. Walker, and W. A. Gray. A framework for automated service composition in service-oriented architecture. In *1st European Semantic Web Symposium*, 2004.

[7] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic matching of Web services capabilities. *Lecture Notes in Computer Science*, 2342:333–347, 2002.

[8] Evren Sirin, Bijan Parsia, and James Hendler. Template-based composition of semantic web services. In *AAAI Fall Symposium on Agents and the Semantic Web*, Virginia, USA, November 2005.

[9] Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. Adding owl-s to uddi, implementation and throughput. In *Workshop on Semantic Web Service and Web Process Composition*, 2004.

[10] K. Sycara, J. Lu, M. Klusch, and S. Widoff. Matchmaking among heterogeneous agents on the internet, 1999.

[11] David Trastour, Claudio Bartolini, and Javier Gonzalez-Castillo. A semantic web approach to service description for matchmaking of services. In *The first Semantic Web Working Symposium, Stanford University, California, USA, July 30 - August 1, 2001SWWS*, pages 447–461, 2001.

[12] Amy Moormann Zaremski and Jeannette M. Wing. Signature matching: a tool for using software libraries. *ACM Transactions on Software Engineering and Methodology*, 4(2):146–170, 1995.

[13] Amy Moormann Zaremski and Jeannette M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.

Authors

Autili, 15

Ben Mokhtar, 137
Bucchiarone, 51

Canfora, 3
Cortellessa, 15

de la Riva, 83
de Vries, 67
Di Marco, 15
Di Penta, 3

Ernst, 123

Frantzen, 67

Garcia-Fanjul, 83
Georgantas, 137
Gnesi, 51
Gonczy, 35

Inverardi, 15
Issarny, 137

Kaul, 137

Lencevicius, 123

Perkins, 123

Smythe, 95

Tretmans, 67
Tuya, 83

Varro, 35