3rd Workshop on Emerging Web Services Technology

Dublin, Ireland
November 12, 2008

Walter Binder
Schahram Dustdar (Eds.)

Prof. Walter Binder
Faculty of Informatics
University of Lugano
Via Buffi 13, CH-6900 Lugano, Switzerland
E-mail: walter.binder@unisi.ch

Prof. Schahram Dustdar
Information Systems Institute
Vienna University of Technology
Argentinierstrasse 8/184-1, A-1040 Vienna, Austria
E-mail: dustdar@infosys.tuwien.ac.at

Emerging Web Services Technology 2008

ECOWS-2008 Workshop, WEWST-2008
Dublin, Ireland, November 12, 2008

http://www.inf.unisi.ch/faculty/binder/wewst08/

Preface

The 3rd Workshop on Emerging Web Services Technology (WEWST-2008) took place in conjunction with the 6th European Conference on Web Services (ECOWS-2008) on November 12, 2008, in Dublin, Ireland.

WEWST focuses on research contributions advancing the state-of-the-art in Web Services technologies. The main goal of the WEWST workshop is to serve as a forum for providing early exposure and feedback to grow and establish original and emerging ideas within the Web Services community. The variety of tools, techniques, and technological solutions presented at WEWST share one common feature: they advance Web Services research in new directions by introducing novel ideas into the field.

We thank the authors for their submissions and the Program Committee for their thorough reviews. We selected 6 full papers and 3 work-in-progress papers for this edition of WEWST. We thank the ECOWS conference organizers for their support without which this event would not have happened.

October 2008

Walter Binder and Schahram Dustdar
WEWST-2008 Program Chairs

Organization

Program Chairs

Walter Binder, University of Lugano, Switzerland
Schahram Dustdar, Vienna University of Technology, Austria

Program Committee

Luciano Baresi, Politecnico di Milano, Italy
Sami Bhiri, DERI Galway, Ireland
Ciarán Bryce, INRIA Rennes, France
Christoph Bussler, Merced Systems, USA
Monique Calisti, Whitestein Technologies, Switzerland
Malu Castellanos, HP, USA
Ion Constantinescu, Digital Optim, USA
Francisco Curbera, IBM T.J. Watson Research Center, USA
Emanuele Della Valle, CEFRIEL, Italy
Elisabetta Di Nitto, Politecnico di Milano, Italy
Peter Dolog, Aalborg University, Denmark
Luis Ferreira Pires, University of Twente, The Netherlands
Walid Gaaloul, DERI Galway, Ireland
Daniela Grigori, University of Versailles, France
Paul Groth, University of Southern California, USA
Thomas Gschwind, IBM Zurich Research Lab, Switzerland
Yanbo Han, Chinese Academy of Sciences, China
Jörg Hoffmann, SAP Research, Germany
Manolis Koubarakis, National and Kapodistrian University of Athens, Greece
Claus Pahl, Dublin City University, Ireland
Cesare Pautasso, University of Lugano, Switzerland
Volker Roth, FX Palo Alto Laboratory, USA
Bernhard Scholz, University of Sydney, Australia
Heiko Schuldt, University of Basel, Switzerland
Niranjan Suri, Institute for Human and Machine Cognition, USA
Stefan Tai, University of Karlsruhe, Germany
Hong-Linh Truong, Vienna University of Technology, Austria
Alex Villazón, University of Lugano, Switzerland
Wolfram Wöß, Johannes Kepler University Linz, Austria
Wolf Zimmermann, University of Halle, Germany

Table of Contents

Full Papers - Morning Session

Full Papers - Afternoon Session

Work-in-progress Papers

Enforcing Advance Reservations for E-Science Workflows in Service Oriented Architectures*

Christoph Langguth and Heiko Schuldt

University of Basel
Department of Computer Science
Database and Information Systems Group
Bernoullistrasse 16
CH-4056 Basel

Abstract. Scientific Workflows have become an important tool to perform complex calculations, especially when individual operations are made available as services in Service Oriented Architectures. At the same time, Quality-of-Service aspects and Advance Reservation of resources by means of Service Level Agreements (SLA) are topics that get ever-increasing attention in order to make best use of available resources in a predictable manner. The support of such SLAs at the level of workflows raises two interrelated issues pertaining (i) to the temporal prediction of reservation start time and duration of individual activities, and (ii) to the actual enforcement of resource commitments at the provider side.
In this paper, we outline our vision of a distributed workflow engine with support for SLAs and Advance Reservations. We focus on reservations addressing processing capabilities, i.e., shares of CPU power. In particular, we present a module of the system that is responsible for the enforcement of such reservations at the individual service providers' nodes, which, by means of a Fuzzy Controller adjusting task priorities, makes sure that the SLAs are met in a fair way.

Key words: Advance Reservation, SOA, Service Grid, Scientific Workflows, CPU share enforcement

1 Introduction

As Service Oriented Architectures (SOAs) are becoming widely deployed in a variety of domains, e.g., in e-Commerce or e-Science, the focus is shifting more and more from mere deployment and integration issues to other, non-functional aspects like Quality of Service (QoS), for example to reserve storage, network or computational resources in advance. A SOA separates functions into distinct units (services), which can be distributed over a network and can be combined and reused to create larger-scale applications (workflows). A widespread

* This work has been partly supported by the Hasler Foundation within the project COSA (Compiling Optimized Service Architectures)

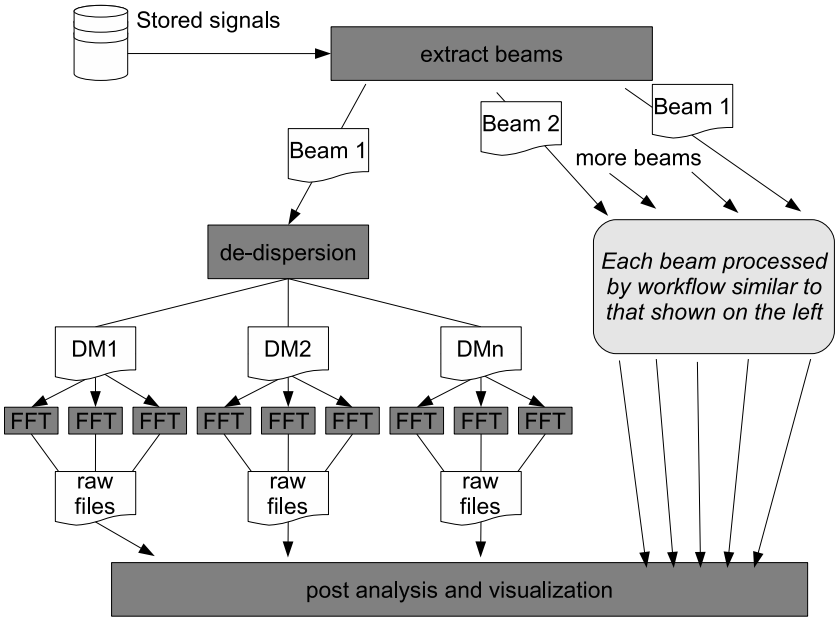


Fig. 1. Pulsar astronomy workflow, according to [4]

language used for defining such workflows in WSDL/SOAP-based SOAs is the Business Process Execution Language (BPEL [2]).

For instance, consider the workflow from the Pulsar Astronomy domain, depicted in Figure 1. This workflow, described in detail in [4], is used to discover and visualize radiation caused by pulsars. For each beam extracted from captured radiation signals, a number of computation steps – each of which can be implemented as a web service (WS) – has to be performed, namely several dedispersion measures, followed by multiple Fast Fourier transforms, and a final aggregation and visualization step. Note that the presented workflow is currently not run in a SOA (but in a traditional cluster environment using MPI), however the authors state that a transition to SOA is envisaged [4].

Another workflow, from the Earth Observation Domain, is described in [6]. These processes feature both attributes that are generally used to distinguish so-called *Scientific Workflows* from Business workflows: vast amounts of data, along with computationally expensive processing steps. While QoS may be a requirement for some applications, e.g., when results are needed in real-time or generally "as fast as possible", any kind of service execution can benefit from the predictability that such QoS contracts (or Service Level Agreements, SLAs) can provide. Assuming that the agreement covers, for example, computational power, i.e., CPU times or shares, service consumers can weigh cost against speed of execution, based on the individual requirements. Service providers may be able to achieve an (economically) optimal resource usage by careful negotiation.

Suppose that a user wants to run the abovementioned workflow taking advantage of QoS criteria, where SLAs with the service providers are established by Advance Reservations (AR). While this task is still relatively easy for individual service invocations scheduled to start at a given point in time, to use ARs in a composed service workflow which consists of a partially ordered set of service invocations, one needs to answer the following two questions:

1. For how long should a reservation for a particular service be made? Since the service implementation is on the provider side, it is generally the service provider that has to make this information available. Note that the provider also has to take measures to enforce this prediction, such as controlling CPU usage.
2. When should a reservation for a particular service start? In a workflow setting, individual service calls will usually depend on the execution or output of previous operations – so anticipating the start time in turn resolves to answering the previous question.

Our objective is to develop and evaluate a system, called DWARFS (Distributed Workflow engine with Advance Reservation Functionality Support), that can support QoS at workflow level. In this paper, we address the first question above, i.e., enforcing CPU usage levels, which is of fundamental importance to proceed with our overall vision. We show that, by using a fuzzy controller to dynamically adjust thread priorities, it is possible to closely confine tasks to the CPU percentage committed to in the SLA, and that one can obtain relatively accurate runtime predictions for future reservations by extrapolating from the runtime and the observations made during the enforcement.

The remainder of this paper is structured as follows: Section 2 shortly introduces the overall DWARFS system. Sections 3 and 4 present our approach to CPU usage enforcement and runtime prediction, as well as an evaluation of first results. Section 5 gives an overview of related work. Finally, Section 6 concludes.

2 Overview of the DWARFS system

The vision of the DWARFS system (Distributed Workflow engine with Advance Reservation Functionality Support) is an advanced BPEL orchestration engine, particularly tailored to e-Science workflows, that is:

– **Fully decentralized.** Whereas any workflow execution is by definition decentralized in the sense that the operations take place at various independent providers, our goal is to also distribute the orchestration engine itself, eliminating the need for a central component controlling the workflow execution. To name just a few assets, a decentralized system helps avoid bottlenecks, hot-spots and single points of failure that a centralized execution engine could potentially create. In addition, especially in scientific workflows where large data volumes are transported during the orchestration, overall performance also may benefit from having the execution engines in proximity to

the target services, by jointly selecting the providers of the workflow's activities and the providers to store instance data in a way that allows to minimize data transfer during workflow execution.
- **WS-Agreement capable.** Ultimately, a workflow execution should be subject to SLAs just like a "normal" WS execution can be. This means that the workflow engine, from a customer perspective, is a service (and agreement) provider, while in essence it is acting as a proxy that itself has to take the customer role for negotiating agreements with the providers of the target services.

For the distributed execution engine part, we can revert to previous experiences from implementing similar systems based on OSIRIS [7, 18], which will be enhanced and extended. For the WS-Agreement (WS-A, [3]) part however, entirely new components have to be developed. This poses a lot of challenging questions including, but not limited to, the (semantic) evaluation of the agreement terms and matchmaking of the possible providers, re-negotiation strategies especially for failure handling, possibly redundant reservation strategies for extremely important processes, etc.

Since we are mostly interested in timing issues (because these are crucial for the agreement establishment for entire workflows), this leads to a natural focus on the *prediction* and therefore *control* of (wall-clock) runtime. As for CPU-intensive tasks, the runtime is directly related to CPU usage, we use a basic model where clients negotiate a reservation for a particular share of the available CPU with a service provider. The service provider gives an estimation of the (maximum) expected runtime for the provision of the service, and its degree of confidence that the estimation will be met. Whereas several implementations for the negotiation of WS-A exist, to our knowledge currently none exists that is able to *enforce* the abovementioned requirements at runtime.

In order to support SLAs at the workflow level, all individual service providers taking part in the workflow must support the respective SLAs as well. As a first step, we therefore developed a component which is able to control CPU usage, restricting it to a given value, and to derive the information needed for a correct runtime estimation. DWARFS uses a Java-based implementation of all components, because of the widespread use of Java in the SOA field and its well-known advantages such as portability and potential for re-use. While we opted for a Java implementation, the approach is not limited to a Java environment. In fact, we show that even in an environment where direct access to the system scheduler is not available, it is possible to accurately control the CPU consumption of tasks. In other environments, the techniques to control the CPU usage may be different, but the conclusions drawn regarding the prediction of future runtimes remain valid – thus, the approach, or a variation of it, can be extended to legacy implementations found in the eScience domain. The basic functionality of the components is as independent as possible from concrete container implementations, thus easing ports of the prototype implementation targeted at a Globus Toolkit 4 (GT4) container. In addition, this component is meant to be as non-invasive as possible: while it may require certain adjustments

to the container or its configuration, it does not require any changes to the OS, the JVM, or – most importantly – the actual service implementations.

Figure 2 presents an enactment scenario where the DWARFS components have been deployed in several of the WS containers providing the target services for the workflow. While the presence of the Process Execution (PE) module is not mandatory on all nodes, as service calls can equally well be made remotely, to deliver the added value of QoS we assume the Advance Reservation (AR) module (or a substitute providing its functionality) to be present.

3 Enforcing CPU share based SLAs

In this section, we describe the architecture and logic of the AR module of the DWARFS system. The module is comprised of the three main components depicted in Figure 3, namely the Agreement agent, which uses previously gathered statistical data to negotiate agreements and authorizes WS-A-bound service invocations; the Supervisor, which monitors the execution of operations (called tasks), enforcing the requested minimum QoS level by accelerating or slowing down execution of individual tasks; and a Fuzzy Controller [9], used for actual decision-making based on a set of configurable rules. In what follows, we summarize our basic assumptions and focus on the two latter components.

3.1 Model and Basic Assumptions

First and foremost, the goal of predicting the execution time of an operation is actually proven to be unachievable in the general case, as it projects to the halting problem [16]. However, assuming that service operations do deterministically provide a result, we argue that a prediction is in many cases possible based on the extrapolation of past results. This leads to the second assumption that such an extrapolation is possible and reasonable, without considering the actual input

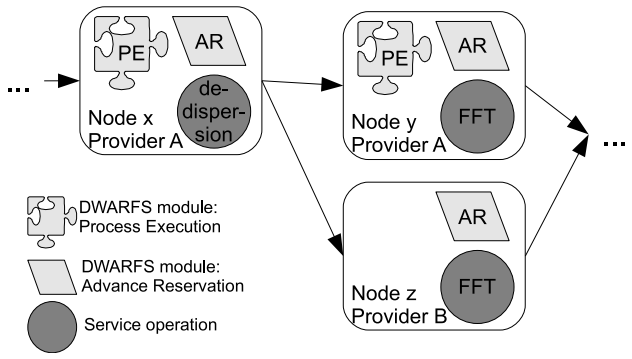


Fig. 2. Sample enactment of a part of the pulsar astronomy workflow using DWARFS

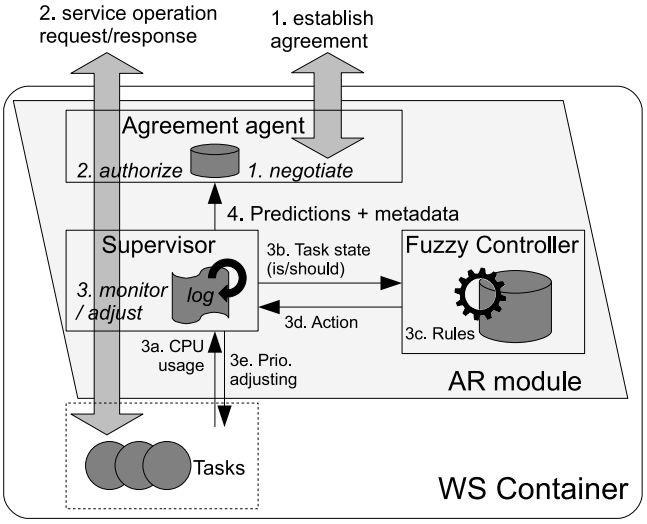


Fig. 3. CPU share controller

data. This might be a limiting factor, and overcoming it or dampening its impact is left for future research. Other factors we do not consider (yet) include the effect of I/O-bound operations (as opposed to the CPU-bound ones we assume), and effects of synchronization and locking in multi-threading calculations.

From a Java program, interactions with the system scheduling are rather limited: to retrieve information about CPU usage, one can only query the number of CPU time all considered threads have used. Similarly, to influence the scheduler, one can only set thread priorities to one of the 10 Java priorities (or, in extreme cases, suspend and resume threads). This results in a rather coarse granularity of possible actions to influence the scheduling.

Our experiments have shown that the actual CPU shares – i.e., the percentage of processing power that threads running at different Java priorities get – are heavily depending on the Operating System scheduler and largely varying between different OS's. Figure 4 shows a representative part of these experiments, where three threads were run in parallel, with the priority of the first thread fixed to 8, and the other two threads taking all possible combinations of priority values from 1 to 8. The resulting CPU shares are depicted textually and graphically, where each thread is represented by a different color. The fact that not all possible requested share combinations can be accomodated by a fixed combination of Java priorities (thus requiring adjustments at runtime), and the rather big differences in behavior of the schedulers among different OS's were the main motivation for using a fuzzy controller to dynamically adjust the priorities at runtime.

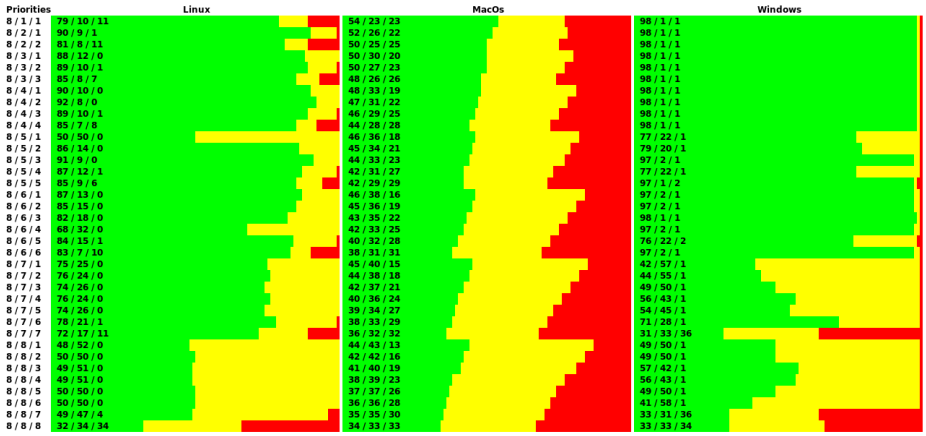


Fig. 4. Mapping of Java thread priorities to effective CPU shares on different OS's

The way of gathering information about CPU usage in Java has another implication: to determine the effective CPU percentage of a thread, one has to sum up all threads' used CPU times to determine the 100% ratio, and then to calculate the actual shares.[1]

Figure 5 depicts this relationship and the overhead introduced by various other parts of the system. Fig. 5 (a) represents 100% of the physical CPU available. In Fig. 5 (b), the overhead introduced by the OS, the JVM, and the AR module itself are depicted. Finally, Fig. 5 (c) shows what the supervisor would see as 100% if the system was not fully loaded – this is caused by the way the calculations are performed, as explained above. However, because we only rely on relative shares, the reasoning is still correct regardless of the actual load on the system. Note that the figure is not drawn to scale, but purely illustrational – while we cannot reliably measure the OS and JVM overhead, we expect them to be rather low, and our measurements have shown that the overhead of the supervisor, in terms of CPU usage, is negligible.

3.2 Maximum Task Share Calculation

Each operation call will result in one or more threads running, which we define as constituting a *task*. With multi-CPU machines, an additional factor has to be taken into account: On a machine with P processors, the maximum achievable share of a task t with n threads is $s^t_{max} = min(1, \frac{n}{P})$. If the system allowed reservations for more than s^t_{max}, the task could not be able to achieve the expected share, resulting in an erroneous slowdown of other simultaneously running tasks. For instance, on a dual-CPU machine with two threads running at full speed, each thread will run on one CPU – a reservation combination of 70%/30% will

[1] Shares are represented as real numbers in the range $[0, 1]$ in the model. However, to ease the understanding, we mostly use the equivalent percentage representation.

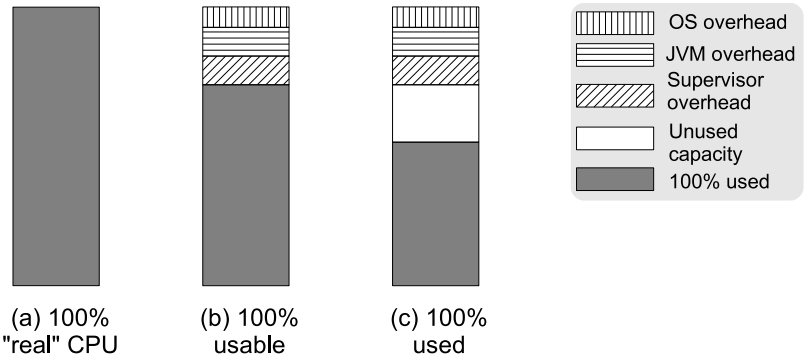


Fig. 5. CPU shares and overhead

result in the first thread never being able to achieve its envisaged goal, but to be blocked at (a maximum of) 50%. The second thread, however, is also not abiding to its 30%, because no matter how low the priority is, the thread will utilize the otherwise unused CPU and run at 50%. It is therefore crucial to know s^t_{max} for a given operation before accepting a reservation request for s^t_{req}, so that these limits can be enforced. This results in the requirement to know the number of threads a given operation will run – which could be provided by the service description, or determined empirically from past executions as well.

3.3 Monitoring and Control of CPU Shares

On task startup, the supervisor gets the necessary metadata (requested CPU share, and number of threads) from the agreement agent. The supervisor, in conjunction with the fuzzy controller, periodically performs the following calculations (labeled *3a* – *3e* in Figure 3) to monitor and control execution for the set T of currently active tasks:

- Calculating the current expected shares (s^t_{cur}) of all tasks, and adjusting them so that $\forall t \in T : s^t_{req} \leq s^t_{cur} \leq s^t_{max} \wedge \sum_{t \in T} s^t_{cur} = 1$. Note that this implies that tasks may get more resources – and thus finish faster – than requested. The objective of the supervisor is to avoid tasks getting too few resources.
- Gathering the CPU usage, and computing the actual share $s^t_{act} \forall t \in T$. So, while s^t_{cur} represents the currently expected share for a task, s^t_{act} is the currently measured share.
- Passing s^t_{act} and s^t_{cur} to the fuzzy controller, and possibly adjusting the thread priority for all of the task's threads in response to the controller output.

Note that the supervisor does not address a "full-fledged" scheduling problem (i.e., it neither has to, nor wants to, assign exactly which tasks have to be run at which moment, which anyway would require to entirely replace the OS or the JVM scheduler). Instead, it merely modifies the priorities of tasks so that their overall CPU consumption matches the requested one.

3.4 Fuzzy Controller Details

The controller used in DWARFS is actually a generic fuzzy controller built for the purpose of, but not limited to usage in, DWARFS. It is completely (re-)configurable at runtime (i.e., all the logic performed, such as getting or setting the system state, fuzzification of parts of it into fuzzy values, fuzzy rule evaluation, and defuzzification, is configured declaratively), and supports detailed logging of the system state. A UI provides users with the ability to perform the configuration, as well as to "replay" and single-step through logs for analyzing them.

In DWARFS, we use 25 rules that evaluate two fuzzy input variables, namely $badness = f(s^t_{act}, s^t_{cur})$, representing the deviation of the actual vs. the expected state, and *tendency*, which reflects the derivation of badness over time. The rule conclusions modify the output variable *action*, which corresponds to the change in thread priorities (-10 to 10) to perform. The following is a textual representation of one of the rules used: `IF badness is overspent_high AND tendency is dropping_slowly THEN action is lower_little`.

3.5 Predicting execution times

After a task has finished, the supervisor aggregates the log information about the elapsed times and CPU usage for the execution and hands this information to the agreement agent, which in turn uses it for future predictions for the operation during agreement negotiation. If task t had run for n intervals with different expected shares (s^t_{cur}), its overall execution E_t can be represented as a set of n time slices $\tau_i = \langle \delta_{\tau_i}, \sigma_{\tau_i} \rangle$, where $\delta_{\tau_i} \in \mathbb{N}^+$ is the duration of the ith slice, and $\sigma_{\tau_i} \in (0, 1]$ is the corresponding actual CPU usage. The predicted execution time for t is then calculated as $PE_t = \frac{1}{s^t_{max}} \sum_{i=1}^{n} \delta_{\tau_i} \sigma_{\tau_i}$. This prediction can be linearly scaled if shares other than s^t_{max} are requested.

4 Evaluation

For evaluation and comparison purposes, we repeatedly (15 times) ran the following configuration: The same CPU-intensive operation (repeatedly calculating SHA-512 hashes, as a representative of a purely CPU-bound and expensive calculation) is run as 6 different tasks, started at different times and with varying requested priorities. This setting was chosen since it contains most of the interesting aspects of a real-life setting, i.e., tasks starting at "random" times (also

at the same time), high-priority tasks intercepting lower-priority ones, tasks acquiring additional (otherwise idle) CPU resources, etc. All tests were performed on the same computer, running on Ubuntu 8.04 (64-bit) and Windows XP SP2 (32-bit), in a normal, not otherwise loaded configuration. In all cases, a Sun JVM 1.6 has been used, and the control loops were effectuated every 500 ms.

Figure 6 depicts the evolution of the system state, as seen from the controller. For each task t, s^t_{cur} (*should*) and s^t_{act} (*is*) are depicted. An important point is that *should*-values are adjusted as tasks join and leave, defining the slice boundaries and resulting in a stair-case-like *should* curve. The controller tries to keep *is* as close to *should* as possible. The oscillations at the boundary start are caused by the fact that each adjustment of the target values (*should*, or s^t_{cur}) results in the need to take the boundary as the new starting point for share calculation, thus starting the calculations "from scratch". Naturally the resulting coarse granularity of input data, paired with few reference intervals, cause a greater imprecision in the calculations and therefore peaks in the representation. In fact, a more intuitive representation of the system state – and more insight into the effectiveness of the controller – is gained by accounting for the performance during previous timeslices, which is done by calculating *as* and *ai* as the average of all *should* (respectively *is*) values over the complete lifetime of the task. These aggregated values are depicted in Figure 7.

Table 1 presents the evaluation of our measurements. The uncontrolled execution time corresponds to the task being run as a standalone application outside of the controller and serves as a control variable. While we cannot explain the striking difference in execution times between Windows and Linux (possibly

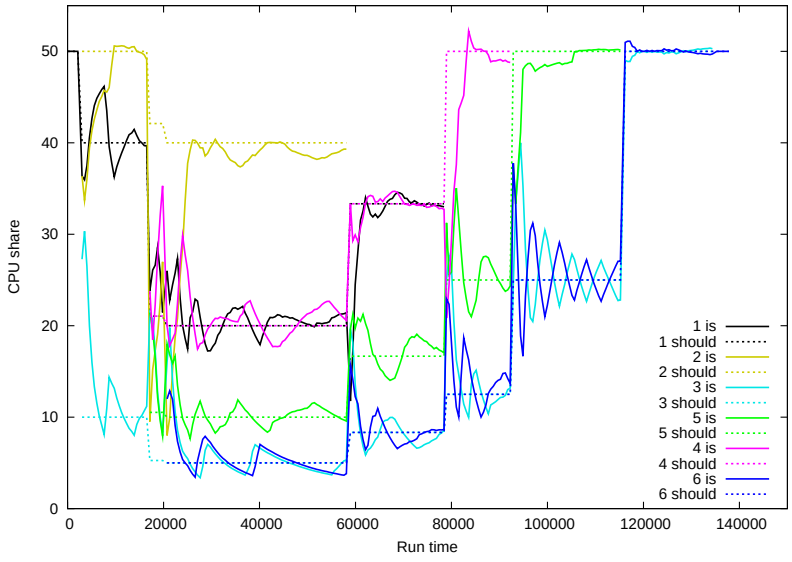


Fig. 6. System state evolution during CPU share controller run

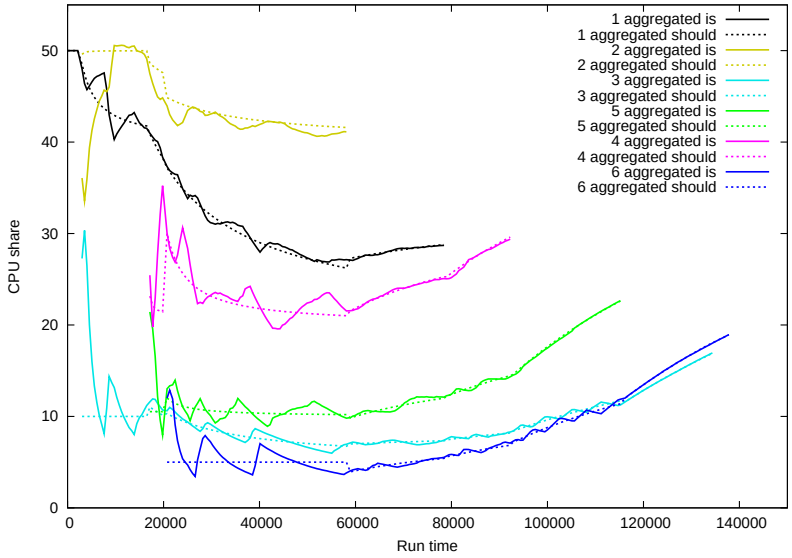


Fig. 7. System state evolution (aggregated shares)

caused by the difference between 32 and 64 bit mode), it is actually helpful for analyzing the effect of longer task run times.

The most important functional quality criteria are the absolute and relative errors, which correspond to an inability of the system to enforce the requested reservations. The results indicate that indeed it is possible to enforce reservations, with the quality of the enforcement and the predictions improving with the duration of a task. The price to pay is a performance penalty, as shown by the predictions. The predictions are generally slower than in the uncontrolled case, as (mostly low-priority) tasks overspending their assigned shares have to repeatedly be suspended so that other tasks meet their target shares. For the sake of reducing this penalty, we tested a configuration that disallowed the suspension of tasks. As shown in the last column, this reduces the overhead, but results in a substantial decrease in quality: most importantly, tasks with higher

Table 1. Evaluation results

Item (averaged over 15 runs)	*Windows*	*Linux*	*Linux (no suspend)*
Uncontrolled execution time (ms)	216829	41361	41361
Coefficient of variation for uncontrolled ex. (%)	0.99	2.19	2.19
Predicted execution time (ms)	221290	48502	44557
Coefficient of variation for prediction (%)	2.05	5.01	2.74
Factor prediction/uncontrolled	1.02	1.17	1.07
Average absolute deviation $\|ai - as\|$ (%)	0.63	0.79	3.26
Average relative error $\frac{\|ai-as\|}{as}$ (%)	5.41	5.96	30.56

priority never achieved their target share, while low priority tasks constantly overspent CPU time.

5 Related Work

Systems targeting the problem of orchestration of resources, in conjuction with QoS criteria, are given a lot of attention predominantly in the Grid community, where the provisioning of resources such as storage or processing capacity is a key aspect. A detailed survey on such systems is presented in [19]. Notably, ASKALON [8] provides a tool set that is focused on measuring, analyzing, and predicting performance aspects of grid applications. The VIOLA project provides support for co-allocation of resources, such as storage, network, and computational resources using SLAs, as described in [13]. Within the GRIDCC project [14], a language for specifying QoS requirements at the level of entire BPEL workflows has been defined. In the context of QoS for workflows, [11] addresses the configuration of the entire system environment, including the dynamic selection and deployment of service instances.

As WS-Agreement seems to emerge as the de-facto standard to describe and negotiate SLAs, some weak points concerning dynamic re-negotiation of agreements (which is particularly relevant for workflows) have been pointed out [1, 17]. [15] proposes modifications to the WS-A specification to support completely dynamic renegotiation of SLAs.

The actual enforcement of the requested QoS criteria – i.e., the assignment of shares of processing power to tasks – is a problem closely related to scheduling, a domain targeted extensively by the Real-Time and Embedded Systems community; an overview of this field is given in [12]. While the DWARFS AR component may well benefit from having an optimized scheduler available at the JVM and/or OS level, the approach of using a Fuzzy Controller on top of the existing scheduler helps us achieve the goal of being both non-invasive (not requiring changes to the underlying system) and flexible (functioning with any kind of underlying OS and JVM scheduler).

Concerning the control mechanisms used to measure and predict CPU usage, the J-RAF framework [5] uses an innovative bytecode instruction counting approach. Whereas this results in accurate measurements, it requires patching of all classes to be executed, and the results are not easily projected to actual wall-clock runtime, which is the target of our work. [10] is targeting the prediction of memory consumption of operations, a topic that, albeit not covered by our approach, would present a useful addition to extend the managed QoS criteria.

6 Summary and Future Work

In this paper, we introduced our DWARFS approach to an AR-supporting decentralized workflow execution engine. In particular, we have presented one of its fundamental modules, which enables us to enforce certain computational QoS

criteria at the scheduler level. While these first evaluation results are encouraging, there are still challenging open questions that require further research. This includes the configuration of the fuzzy controller, namely the calculation of the variables and the rulesets, and possibly the evaluation of alternative strategies for controlling combinations of tasks, instead of individual tasks only. Further work will then re-consider the limiting basic assumptions, their practical relevance and possible ways to overcome them or to minimize their impact. At the same time, the implementation and integration of the system will be carried on, so that the focus can be shifted to the larger-scale problems of AR strategies at the level of entire workflows in a distributed setting.

References

1. M. Aiello, G. Frankova, and D. Malfatti. What's in an Agreement? An Analysis and an Extension of WS-Agreement. In *ICSOC*, pages 424–436, 2005.
2. A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guízar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, April 2007.
3. A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). http://www.ogf.org/pipermail/graap-wg/2006-July/000457.html, July 2006.
4. J. Brooke, S. Pickles, P. Carr, and M. Kramer. Workflows in Pulsar Astronomy. In *Workflows for e-Science*, pages 60–79. Springer London, 2007.
5. A. Camesi, J. Hulaas, and W. Binder. Continuous Bytecode Instruction Counting for CPU Consumption Estimation. In *QEST '06: Proceedings of the 3rd international conference on the Quantitative Evaluation of Systems*, pages 19–30, Washington, DC, USA, 2006. IEEE Computer Society.
6. L. Candela, F. Akal, H. Avancini, D. Castelli, L. Fusco, V. Guidetti, C. Langguth, A. Manzi, P. Pagano, H. Schuldt, M. Simi, M. Springmann, and L. Voicu. DILIGENT: integrating digital library and Grid technologies for a new Earth observation research infrastructure. *Int. J. on Digital Libraries*, 7(1-2):59–80, 2007.
7. L. Candela, D. Castelli, C. Langguth, P. Pagano, H. Schuldt, M. Simi, and L. Voicu. On-Demand Service Deployment and Process Support in e-Science DLs: the Diligent Experience. In *DLSci06*, pages 37–51, 2006.
8. T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. S. Jr., and H. L. Truong. ASKALON: a tool set for cluster and Grid computing. *Concurrency - Practice and Experience*, 17(2-4):143–169, 2005.
9. B. R. Gaines. Fuzzy reasoning and the logics of uncertainty. In *Proceedings of the sixth international symposium on Multiple-valued logic*, pages 179–188, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.
10. O. Gheorghioiu. Statically Determining Memory Consumption of Real-Time Java Threads. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2002.
11. M. Gillmann, G. Weikum, and W. Wonner. Workflow Management with Service Quality Guarantees. In *In Proceedings of the 2002 ACM SIGMOD Int. Conference*

on Management of Data, pages 228–239, Madison, Wisconsin, June 2002. ACM Press.
12. J. Leung, L. Kelly, and J. H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., Boca Raton, FL, USA, 2004.
13. H. Ludwig, T. Nakata, O. Wäldrich, P. Wieder, and W. Ziegler. Reliable Orchestration of Resources using WS-Agreement. In *HPCC*, pages 753–762, 2006.
14. A. S. McGough, A. Akram, L. Guo, M. Krznaric, L. Dickens, D. Colling, J. Martyniak, R. Powell, P. Kyberd, and C. Kotsokalis. GRIDCC: real-time workflow system. In *WORKS '07: Proceedings of the 2nd workshop on Workflows in support of large-scale science*, pages 3–12, New York, NY, USA, 2007. ACM.
15. G. D. Modica, V. Regalbuto, O. Tomarchio, and L. Vita. Dynamic re-negotiations of SLA in service composition scenarios. In *EUROMICRO-SEAA*, pages 359–366, 2007.
16. P. P. Puschner and C. Koza. Calculating the Maximum Execution Time of Real-Time Programs. *Real-Time Systems*, 1(2):159–176, 1989.
17. R. Sakellariou and V. Yarmolenko. On the Flexibility of WS-Agreement for Job Submission. In *MGC '05: Proceedings of the 3rd international workshop on Middleware for grid computing*, pages 1–6, New York, NY, USA, 2005. ACM.
18. C. Schuler, C. Türker, H.-J. Schek, R. Weber, and H. Schuldt. Scalable peer-to-peer process management. *Int. J. of Business Process Integration and Management*, 1:129–142(14), 8 June 2006.
19. J. Seidel, O. Wäldrich, P. Wieder, R. Yahyapour, and W. Ziegler. Using SLA for Resource Management and Scheduling - A Survey. In *Grid Middleware and Services - Challenges and Solutions*, CoreGRID Series. Springer, 2008. Also published as CoreGRID Technical Report TR-0096.

Towards Pattern-Based Service Identification

Veronica Gacitua-Decar and Claus Pahl

School of Computing, Dublin City University, Dublin 9, Ireland
vgacitua|cpahl@computing.dcu.ie

Abstract. Service-Oriented Architecture is a promising architectural approach to solve the integration problem originated by business process integration and automation requirements. The identification of the adequate services for the service architecture solution is a critical issue. Architecture abstractions, such as patterns, can capture design knowledge and allow the reuse of successful applied designs. The continual rise of abstraction in software engineering approaches have been a central driver of this work, placing the notion of patterns at business domain level. In this paper we propose a set pattern-based techniques for service identification. Graph-based pattern matching and pattern discovery are proposed to recommend the scope and granularity of services on process-centric description models. Matching of generalised patterns and hierarchical matching are discussed.

1 Introduction

Nowadays, evermore organizations are taking advantage of consolidating relations with service provider companies in order to improve competitiveness. This involves the merging of internal processes from provided and provider companies into inter-organisational processes shaped by a business chain value [1]. At technical level, business process integration creates an Enterprise Application Integration (EAI) problem.

Service-Oriented Architecture (SOA) is a promising architectural approach to solve the EAI problem. The definition of the services that will be the building blocks of the architecture solution is a critical issue. *Abstraction* is a principle that can address this challenge. Architecture abstractions like patterns and styles can capture design knowledge and allow the reuse of successfully applied designs and improve the quality of software [2]. Abstraction in software engineering approaches is a central driver; at the business level the reuse of successfully business designs is equally important.

Service identification is a central activity during the design of service architecture solutions. It involves the analysis of business models and their relation with the existing software support [3]. Existing software support might be implemented as services, or most frequently, as legacy applications. Thus, service identification might involve the *discovery* of existent services, adaptation of those services, or the *definition of new services*. Regarding service discovery -ideally- service requesters can find completely compatible services. However, in some practical scenarios, services that partially fulfill a request might also be of interest. A black box view of services is not sufficient, and considering structural and behavioural information beyond the signatures and effects of services can support a more flexible service discovery. Reuse of services within the limits of one

organisation and its partners, providers and clients in close cooperation can be potentiated by planning in advance the services that will be available. In this manner, *reuse* of services is emphasised at design time - before implementation. This is specially relevant for large organisations where overlapping functionality offered by different services can rapidly grow, overshadowing the benefits of service reuse.

A number of contributions have addressed the problem of service identification. High level guidelines for the design of new services such as in [3] are useful, however they lacks of formality and techniques promoting automation that can be finally materialised as tool support. More specific approaches for service discovery, for example, based on matching of process-centric service descriptions such as in [4],[5],[6] goes in the line of automating the service discovery process. Architecture abstractions in the form of patterns has been exploited at technical level to improve the quality of software [2]. Less explored is the use of patterns at business level and their subsequent refinement to more technical levels. In this paper we present a set of pattern-based techniques and algorithms that focus on the identification of boundaries on process-centric models that recommend the scope and granularity of new services. Note that service discovery has not directly addressed here, however the proposed algorithms and related concepts could contribute to graph-based techniques for exact and partial service matching such as for example the work in [7].

- The definition of *new business-centric services* is addressed by means of *structural matching* between process patterns and process models. *Hierarchical* matching allows incremental levels of abstraction of process-centric matched patterns. Controlled vocabulary of business domains is considered by the matching of *generalised patterns*. *Partial* pattern matching provides flexibility to the proposed techniques.
- The other technique presented here, exploits the fundamental principle of *reuse* in software design. The intuitive idea is to find *frequent* process substructures -named *utility patterns*- within large process models. Process steps related with discovered utility patterns might be supported by existing software components, which can be rationalised, and subsequently encapsulated as reusable technical-centric services.

The remainder of this paper is organised as follows. Section 2 introduces a graph-based representation of process models and its relation with process patterns. Section 3 describes the different aspects of the process pattern matching problem and our proposed solutions. Section 4 describes our initial proposal for finding utility patterns in process models. Section 5 provides a preliminary evaluation of the proposed exact and partial pattern matching techniques. Finally, in sections 6 and 7 a review of the related work and conclusions are provided.

2 Graph-based representation of Business Process Models and Business Process Patterns

Graphs emerge as a natural representation of process-centric models [8],[9]. Graphs can capture both structure and behaviour, and allow abstractions such as patterns to be related to process-centric models .

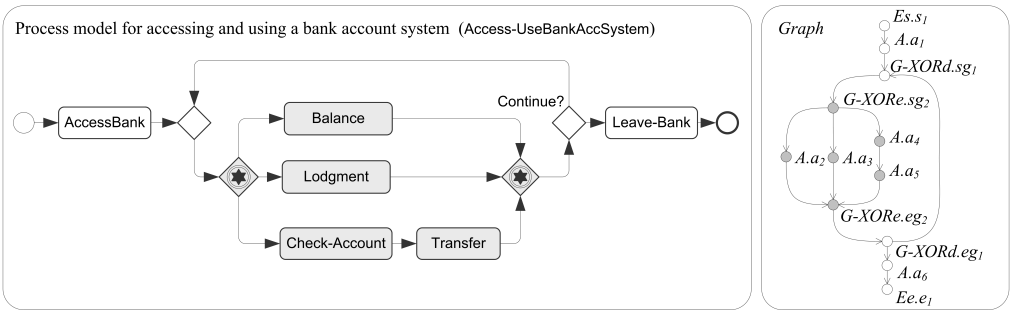


Fig. 1. Process model annotated with BPMN and a related graph-based representation.

2.1 Structural Representation of Business Process Models as Graphs

In the context of this paper we use graphs to represent the structure of process models and process patterns. Graph vertices represent process elements such as activities, control flow elements, and so on. Graph edges represent the connectivity between process elements. Section 8 (annex) provides an introductory background on graphs and related notation that is used in this section and referred to the rest of the paper.

Graph-based business process model. Let the graph $PM = (V_{PM}, E_{PM}, \ell_{V_{PM}}, \ell_{E_{PM}})$ be a finite, connected, directed, labelled graph representing a business process model. V_{PM} is the set of vertices representing process elements and E_{PM} is the set of edges representing *connectivity* between process elements. The function $\ell_{V_{PM}} : V_{PM} \rightarrow L_{V_{PM}}$ is the function providing labels to vertices of PM, and $\ell_{E_{PM}} : E_{PM} \rightarrow L_{E_{PM}}$ is the function providing labels to edges of PM. $L_{V_{PM}}$ and $L_{E_{PM}}$ are the sets of labels for vertices and edges, respectively.

Note that in this paper *connectivity* between process elements is simplified by considering only the sequence flows between activities since we focus on *structural* matching of patterns on process. A more complete approach could capture on edges: inputs, outputs, pre and post conditions regarding execution of activities. The Fig. 1 provides an example of a intuitive graph-based representation of a business process model annotated with a well-known process modelling notation, i.e. Business Process Modelling Notation[1] (BPMN). An appropriate mapping function maps descriptions of process elements with graph labels. Note that similar graph-based models can represent executable processes described for instance in the standard WS-BPEL language[2].

2.2 Structural Representation of Business Process Patterns as Graphs

Business Process (BP) patterns are essentially common connectivity patterns in process models. BP patterns can be operator-oriented, e.g. a multi-choice pattern that allows the selection of a number of options instead of an exclusive selection based on the basic choice operator. These kind of process patterns are know in the literature as *workflow patterns* [10]. Other category of BP patterns consists of application context-oriented and

[1] Available from http://www.bpmn.org/Documents/BPMN 1-1 Specification.pdf
[2] Available from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

often more complex patterns derived from and specific to the business context. These kind of BP patterns can represent well-known process building blocks in reference models, abstracting a set of connected activities required to reach some business goal [11]. Application context-oriented business process patterns can be reused as previously implemented and successful designs and provide an integrated vision of processes among different participants. For instance, in Fig. 1 the Use-AccessBankAccSystem process has at its core, in gray colored vertices, a common set of account usage activities that can be represented in the form of a application-context oriented process pattern.

Beyond the previous types of BP patterns, a third category represent frequent process connectivity structures that are not specific to a business domain, but relates to some standard technology solution, for instance a typical authentication and authorisation processes to access a system. We named this patterns as *utility* patterns, borrowing the name from the definition of *utility* services in [3]. In the rest of the paper we will refer to *application context-oriented business process patterns* only as *patterns*. *Workflow patterns* are not addressed here. *Utility patterns* are the focus of section 4.

Graph-based business process pattern. Let the graph $PP = (V_{PP}, E_{PP}, \ell_{V_{PP}}, \ell_{E_{PP}})$ be the finite, connected, directed, labelled graph representing a business process pattern model. Elements of V_{PP} represent process pattern roles and elements in E_{PP} represent connectivity between pattern roles. Note that the graph-based representation for business patterns, utility patterns and business processes is structurally the same.

2.3 Instantiation of Process Patterns in Process Models

Process patterns have been described in the same way as process models. Now, we discuss the relation between process patterns and process models. In particular, we are interested in the abstraction that patterns represent for process models and concretely, in the notion of *instantiation* of process patterns in process models. *Pattern Instantiation* in a concrete model indicates that the structural relations described in the pattern hold in the model. The structural preserving relations that graph homomorphisms represent help us to capture the notion of pattern instantiation. In particular, instantiation of a BP pattern in a process model can be captured by the definition of a *locally surjective graph homomorphism* [12] between a subgraph PM_S of the graph process model PM and the pattern graph PP, i.e. $PM_S \xrightarrow{S} PP$. *Surjection* allows that several process elements (vertices of PM) play the role of one pattern element (vertex of PP). Moreover, model elements can belong to more than one pattern when considering this approach. Note that we have used the notation from the previous section and the Annex. We will continue using this notation along the paper.

3 Process Pattern Matching

We have discussed in Section 1 the potential that discovering instances of patterns in concrete models can provide to the definition of new services. Matching a pattern in a concrete model involves the identification of instances of that pattern in the concrete model. In this manner, the *pattern matching problem* can be referred as the detection of a

graph homomorphism between the graph representing a concrete model and the graph representing the pattern.

3.1 Exact, Inexact and Partial Pattern Matching

In realistic scenarios where an *exact* match of a pattern is unlikely, *partial* and *inexact* matching become relevant. *Inexact pattern matching* provides *good*, but not exact solutions to the matching problem. In this case, pattern instances can incorporate additional elements not described in the pattern, nevertheless they must not affect the structural properties of the pattern. *Partial pattern matches* identify exact but *incomplete* matches of patterns. Partial instances of patterns might exist due to a modification or evolution of a previously instantiated pattern. However, when patterns have not previously considered as part of the design, partial matches indicate an opportunity to improve the design through incorporating the whole pattern. *Partial* and *inexact* matches are also important due to the fact that process models and their implementations as services might be highly similar but not exactly the same from organisation to organisation and to identify commonalities can save costs and encourage reuse.

In order to formalise and later on implement our proposed techniques as concrete tool support we will define exact, partial and inexact pattern matching in terms of the graphs representing processes and patterns and their structural relations. Formalisation can provide guaranties of correctness and improve the confidence on tools.

Exact Pattern Matching. A exact pattern match of a specific pattern PP in an arbitrary process model PM refers to the detection of $PM_S \xrightarrow{S} PP$ with $PM_S \subseteq PM$. The mapping function φ defines an individual instantiation of the pattern PP in the process model PM with $\varphi : V_{PM_S} \rightarrow V_{PP}$ satisfying that for all $u \in V_{PM_S} : \varphi(N_{PM_S}(u)) = N_{PP}(\varphi(u))$ and with mapping $\lambda_S : L_{V_{PM_S}} \rightarrow L_{V_{PP}}$ a bijective function indicating a semantic correspondence between the labels of two mapped vertices.

Partial pattern matching. Partial matches restrict the matching problem allowing incomplete matches. Incomplete pattern matches maps elements from PM to a reduced number of elements considered in the original codomain (V_{PP}). In this manner, the original function φ defined by the exact matching case is now restricted to the function $\varphi_{PARTIAL} : V_{PM_{S^*}} \rightarrow V_{PP_{PARTIAL}}$ satisfying that for all $u \in V_{PM_{S^*}}$:
$\varphi_{PARTIAL}(N_{PM_{S^*}}(u)) = N_{PP_{PARTIAL}}(\varphi_{PARTIAL}(u))$ with $PP_{PARTIAL} \subseteq PP$ and $PM_{S^*} \subseteq PM_S$.

Inexact pattern matching. Inexact pattern matching relaxes the definition of *neighborhood* in the Annex (Section 8) by a set $N^*_{PM_S}(u)$ allowing other vertices not only in the neighborhood of a vertex u ($N_{PM_S}(u)$) but also in the *path* between u and v with $\varphi(u)$ adjacent with $\varphi(v)$ and $\varphi : V_{PM_S} \rightarrow V_{PP}$.

Algorithm for Exact and Partial Matching. We propose an algorithm for exact and partial process pattern matching. The pseudo-code of the proposed algorithm is described in **ALGORITHM 1** (uEP-PMA). The algorithm starts matching each vertex in V_{PP} with vertices in V_{PM} such that the labels in $L_{V_{PP}}$ are semantically correspondent with labels in

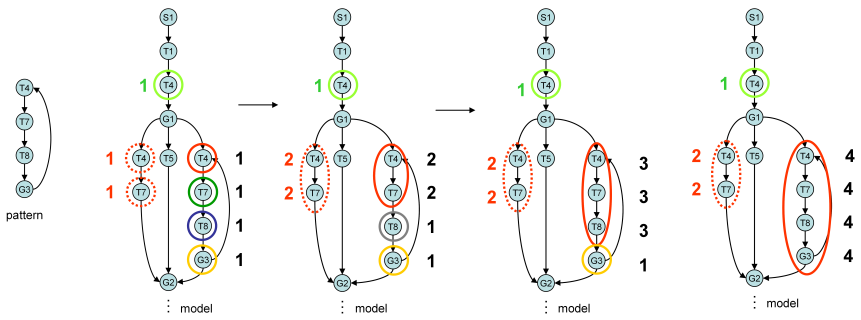


Fig. 2. Matching expansion steps. One exact match and two partial matches are found.

$L_{V_{PM}}$. Semantic correspondence in *u*EP-PMA refers to a one to one (bijective) mapping λ between a subset in $L_{V_{PM}}$ of giving labels to matched vertices in V_{PM} and labels in $L_{V_{PP}}$. Each initial match is considered a temporal pattern matching defining a temporal subgraph in PM that we denote as tPM. Subsequently, tPM is expanded until all its neighbors that hold a structural relation defined by φ or at least $\varphi_{PARTIAL}$ are added.

Fig. 2 illustrates the expansion steps. The algorithm terminates when no more expansion steps can be done. The result is a *score* vector. Each vertex in PM has a score that indicates the number of vertices of the matched pattern to which it belongs.

ALGORITHM 1: *u*EP-PMA - *u*ndirected Exact and Partial - Pattern Matching Algorithm.

Input: Target Graph (PM), Pattern Graph (PP)
Output: Score Vector (*score*).

1 : **For each** vertex m in V_{PM} **do**
2 : **For each** vertex p in V_{PP} **do**
3 : **If** $\lambda \circ \ell_{V_{PM}}(m) = \ell_{V_{PP}}(p)$ == true **then**
4 : $tPM(m) \leftarrow$ initial temporal match centred in vertex $m \in PM$
5 : *score* $\leftarrow 1$ (*score* for vertices in $tPM(m)$)
6 : **end if**
7 : **end for**
8 : **end for**
9 : **Do while** *ExpansionCondition* == true
10 : **For each** vertex $i \in tPM(m)$ **do**
11 : **If** $\ell^{-1}_{V_{PP}} \circ \lambda \circ \ell_{V_{PM}}(N_{tPM(m)}(i)) = N_{PP}(\ell^{-1}_{V_{PP}} \circ \lambda \circ \ell_{V_{PM}}(i))$ && $N_{tPM(m)}(i) \notin tPM(m)$ **then**
12 : Expand $tPM(m)$ with $N_{tPM(m)}(i)$
13 : *score* $\leftarrow$ *score* + 1
14 : *ExpansionCondition* $\leftarrow$ true
15 : **Else if** *ExpansionCondition* $\leftarrow$ false **end else if**
16 : **end if**
17 : **end for**
18 : **end do while**

Note that several exact or partial instances of PP in PM might exist. If different pattern instances share edges in PM, we say that there are *overlaps* of the pattern PP in PM. The *u*EP-PMA algorithm identifies the connected subgraphs in PM containing overlaps

as a one single subgraph PM_O. The score of the vertices in the overlap is the number of vertices in PM_O. Additionally, in order to consider the directionality of the graphs representing concrete models and patterns the *u*EP-PMA algorithm can also be performed on the *undirected* version of *PM* and *PP*. In this manner, matches not only considers vertices, but also arcs.

According to [13], for a connected simple graph H, the problem of detecting a locally surjective homomorphism between an arbitrary graph and H is solvable in polynomial time if and only if H has at most two vertices. In all other cases the problem is NP-complete. The complexity of the latter problem, which is directly related with the pattern matching problem, made us aware of performance issues. In Section 5 we show a preliminary evaluation where instances of specific graph patterns are identified on arbitrary random graphs. The results show that the time required to solve the problem is quadratic in relation to the size of the random graphs and it has a small constant that conveniently modulates the response time for small and medium size graphs. Scalability, in terms of processing several patterns over one or more target graphs, could be addressed by implementing a refined version of the algorithms to allow parallel processing of each pattern to be matched on a target model.

3.2 Matching of Generalised Patterns

Consideration of restricted vocabulary for different vertical business domains can add additional benefits for the practical use of BP pattern matching solutions. There are cases where descriptions of process elements (or pattern elements) have the same syntax, but different semantic and vice versa. Moreover, processes and patterns might be described with different structures, while they behave in the same way. Regarding the vocabulary used to describe process and pattern elements, we have extended the *u*EP-PMA algorithm with the *u*G-PMA algorithm allowing semantic correspondence beyond the one to one mapping (λ) previously considered. The structure of the algorithm remains relatively invariant, but the functions $\ell_{V_{PM}}$, $\ell_{V_{PP}}$ and λ are modified. In this case, the two $\ell_{(\cdot)}$ functions are mapping vertices from *PM* to labels that are organised in a tree-like structured taxonomy. The labels in the taxonomy refer to concepts from a particular business domain. In this manner, *generalised patterns* are considered as families of patterns where the parent pattern contains the roots of tree-structured taxonomies for business concepts. Child patterns contains one or more child concepts connected to root concepts in the hierarchy defined by the taxonomy. Note that using the *u*G-PMA algorithm requires the existence of an implemented taxonomy from where the algorithm can search for semantically correspondent terms.

3.3 Hierarchical Pattern Matching

In the previous sections we have addressed the exact and partial matching problem on flat process models (and patterns). However, processes and patterns are commonly composed by more fine-grained process-centric structures. In this section we outline a solution to the problem of pattern matching considering different levels of abstraction.

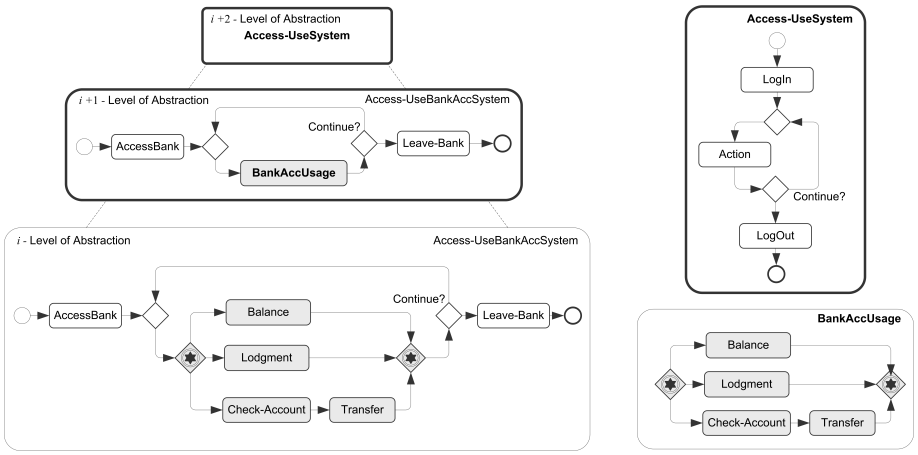


Fig. 3. Hierarchical pattern matching.

Algorithm for hierarchical pattern matching. The pseudo code of the proposed algorithm named uH-PMA is described in **ALGORITHM 2**. The algorithm starts matching at a certain level of granularity on a target model PM different patterns PP_j from a set of patterns $setPP$. The index j identifies an specific pattern in $setPP$. Subsequently, PM is transformed to an abstracted representation PM^i where i represent a particular level of abstraction. Subsequently, subgraphs of PM^i that have been matched with any PP_j are replaced by vertices p_j of type *pattern*[3] in the graph one level of abstraction up (PM^{i+1}). Thus, the complexity of a matched subgraph is hidden in a *pattern vertex* p_j. Note that representative labels are assigned to pattern vertices. Once the target model is abstracted with pattern vertices from matched patterns at a specific level of abstraction, other patterns at a higher level might appear. In this way, the abstraction process can be performed iteratively, abstracting a process graph PM^i into a process graph PM^{i+1} which is one level of abstraction up, and so on. The algorithm terminates when no more matches are found or when the process graph has only one vertex.

The Fig. 3 illustrates the idea of hierarchical pattern matching making use of the process model from Fig. 1. The Fig. 3 shows two patterns: BankAccUsage and Access-UseSystem that are consecutively matched in two different levels of abstraction. The pattern BankAccUsage describes a set of common bank account usage activities and the pattern Access-UseSystem represents a typical -simplified- set of steps to access a generic system. Note that BankAccUsage is focused on the banking industry, however the Access-UseSystem pattern can be valid across different industries since it has a technology-oriented and business-agnostic nature [11]. The first match involves a mapping from elements in the process model Access-UseBankAccSystem to elements in the BankAccUsage pattern. The resultant abstracted process model is subsequently matched with the Access-UseSystem pattern. In this case, checking semantic correspondence is enhanced by using

[3] Typed graphs are graphs that holds a complete mapping to a set of types. Mappings for typed graphs can consider vertices and edges. The mapping function considered for pattern vertices is a global surjective function from the set of graph vertices to the set of types.

a taxonomy for business concepts. The result of the hierarchical pattern matching process is a single vertex referring to the access and use of a system.

Note that we have not addressed the problem of *overlaps* yet. How to abstract two matches that share vertices and edges in the target model? Our basic representation of processes and patterns as graphs restricts the possibility of representing two overlapped matched patterns as two different pattern vertices. One idea that we will explore is the representation of matched patterns as hyperedges of a hypergraph. The vertices of the hypergraph are the same vertices of the graph representing the process model.

ALGORITHM 2: *u*H-PMA - *u*ndirected Hierarchical - Pattern Matching Algorithm.

Input: Target Graph (*PM*), Set *setPP* of *n* pattern graphs ($setPP = \{PP_1, ..., PP_n\}$)
Output: *scoreMatrix*[4].

```
1 : Do while IterationCondition&&change == true
2 :   For each pattern PP_j ∈ setPP do
3 :     uEP-PMA(PM^i, PP_j) (or uG-PMA if generalised pattern matching is desired)
4 :     If score(u) = |V_PPj| with u ∈ PM^i_Sj && exact match == true then
5 :       PM^i_Sj ← p_j
5 :       change ← true
5 :       If |V_PM^i| <= 1 then
5 :         IterationCondition ← false
5 :       end if
6 :     end if
6 :     Else if
6 :       change ← false
6 :       i ← i + 1
7 :   end for
8 : end do while
```

4 Discovering Frequent Utility Patterns in Process Models

Previous sections described techniques for identifying services based on the matching of known application context-oriented process patterns in process models. In this section we are interested in discovering frequently occurring substructures on large scale business process models. Process steps might be supported by existing software components and identifying reoccurring connected process steps provide a medium to define potential reusable software components as encapsulated services. The idea is to exploit the basic principle of reuse in SOA. Finding frequent -not necessarily known- *utility patterns* in large process models can help to the definition of reusable technical-centric services.

There are two distinct problem formulations for frequent pattern discovery in graphs: graph-transaction setting and single-graph setting [14]. The latter refers to the discovery of subgraphs that occur multiple times in a single input graph. The other refers to the

[4] *scoreMatrix* is a matrix where each element is a *score* vector derived from algorithm *u*EP-PMA. Rows in *scoreMatrix* refer to the level of granularity *i* of the model *PM* and the columns refer to the different matched patterns $PP_j \in setPP$.

discovery of subgraphs that occur frequently across a set of small graphs. We present an algorithm focused on single-graph setting scenario for pattern discovery in graphs.

Algorithm for Pattern Discovery. The algorithm attempts to find frequent -exact and partial- occurrences of subgraphs in a single input graph PM. A discovered frequent subgraph -utility pattern- is an induced subgraph PP_U homomorphic with all occurrences of a frequent subgraph of PM. Homomorphism detection in the proposed algorithm (named uEP-FPDA) relies on the pattern matching algorithm uEP-PMA described in Section 3.1. The pseudo code of uEP-FPDA is described in ALGORITHM 3.

ALGORITHM 3: uEP-FPDA - $u$ndirected Exact and Partial - Frequent Pattern Discovery Algorithm.

Input: Target Graph - undirected version (uPM), Threshold (Th), number of expansion steps (k)
Output: $score$, $FreqM$

1 : **For each** vertex u in uPM **do**
2 : $PP_{pivot(u,1)} \leftarrow u$
3 : $seeds_{(u,1)} \leftarrow u$EP-PMA$(PM, PP_{pivot(u,1)})$
4 : $score_{(u,1)} \leftarrow seeds_{(u,1)}$
5 : **For each** i in $seeds_{(u,1)}$ **do**
6 : **If** $score_{(u,1)}(i)/|PP_{pivot(u,1)}| >= Th$ **then**
7 : $cnt_{(u,1)} \leftarrow cnt_{(u,1)} + 1$
8 : **end if**
9 : **end for**
10 : $FreqM(u,1) \leftarrow cnt_{(u,1)}/|PP_{pivot(u,1)}|$
11 : **If** $k >= 1$ **do**
12 : **For** $j : 2 \rightarrow k$
13 : $PP_{pivot(u,j)} \leftarrow expand(PP_{pivot(u,j-1)})$
14 : $seeds_{(u,j)} \leftarrow u$EP-PMA$(PM, PP_{pivot(u,j)})$
15 : $score_{(u,j)} \leftarrow seeds_{(u,j)}$
16 : **For each** i in $seeds_{(u,j)}$ **do**
17 : **If** $score_{(1)}(u,j)/|PP_{pivot(u,j)}| >= Th$ **then**
18 : $cnt_{(u,j)} \leftarrow cnt_{(u,j)} + 1$
19 : **end if**
20 : **end for**
21 : $FreqM(u,j) \leftarrow cnt_{(u,j)}/|PP_{pivot(u,j)}|$
22 : **end for**
23 : **end if**
24 : **end for**

The size of the induced subgraphs and a parameter that relaxes the way of counting the frequency of occurrences of induced subgraphs are parameterised in k and Th, respectively. The constant k refers to the amount of times that an initial arbitrary subgraph in PM will be *expanded* and compared with other subgraphs in PM to check for homomorphisms. Th refers to a threshold for the ratio between the number of vertices of two non exact occurrences of PP_U. If Th is equal to one, the frequent occurrences of subgraphs in PM must to be isomorphic between them. The output of uEP-FPDA are two matrices $score$ and $FreqM$. In the matrix $score$ rows represent each vertex u in PM and columns the results for different size of pattern. If u belongs to a highly frequent

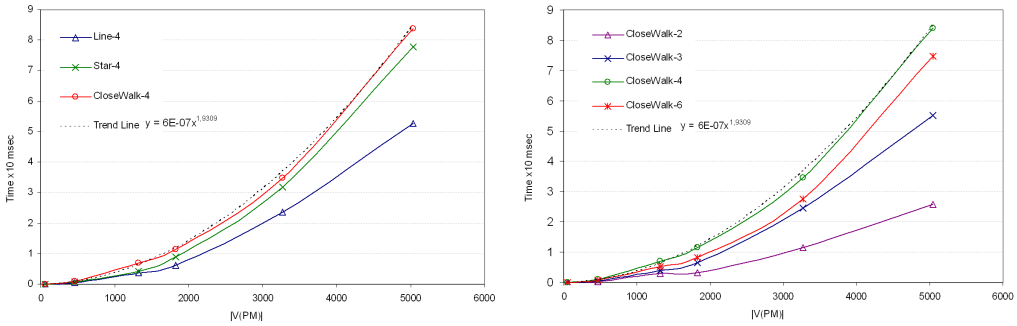


Fig. 4. Average response time of uEP-PMA on arbitrary random graphs for different pattern structures (left side) and different pattern sizes (right side).

subgraph in PM of size j then $score(u, j)$ will be also high. $FreqM$ is a matrix with $|V_{PM}|$ rows and k columns, where each cell indicates the frequency of a discovered pattern centred the vertex indicated by the row and with size indicated by k.

The uEP-FPDA algorithm starts defining an arbitrary vertex u from the target graph PM as the first temporal pattern (pivot pattern or $PP_{pivot(u,1)}$) and matching $PP_{pivot(u,1)}$ with the rest of the target graph. The matrices $score$ and $FreqM$ are initialised with the results of the matching for the initial pattern of size 1. The next steps are repeated for each vertex in PM. The subgraph $PP_{pivot(u,1)}$ is expanded with its neighbors, together with expanding each of the vertices in PM whose were matched with the initial $PP_{pivot(u,1)}$. These first matched vertices are called $seeds_{(u,1)}$. The algorithm continues the expansion of PP_{pivot} while checking if there exist an homomorphism between the expanded PP_{pivot} and subgraphs in PM. The counting for measuring the frequency of the matched subgraphs -expanded $seeds$- depends on the satisfaction of the threshold Th parameter. The expansion process continues until k times or no more homomorphisms are detected. The results contained in $score$ and $FreqM$ indicates the set of induced subgraphs PP_U -discovered utility patterns- centred in the initial $seeds$ and the PP_{pivot}.

Based on the results obtained in the preliminary evaluation (Section 5) indicating the quadratic complexity order of uEP-PMA, it is expected that for uEP-FPDA the complexity order grown up to $O(kV^3)$, where V the size of the problem in term of the number of vertices and k the number of times the temporal patterns in uEP-FPDA is expanded.

5 Evaluation

We have performed a preliminary evaluation for the exact and partial matching algorithm (uEP-PMA). The experiments consider seven specific patterns over arbitrary random graphs with approximate sizes of 60, 450, 1300, 1800, 3200 and 5000 vertices. The experiments were run in a Intel machine 2 GHz and 2GB RAM on WinXP-SP3. Labels in patterns and random graphs can be of three different types: A, B or C. The used patterns are a four close-walk of 2, 3, 4 and 6 vertices; two line-like patterns of 3 and 4 vertices and a star-like pattern with 4 vertices.

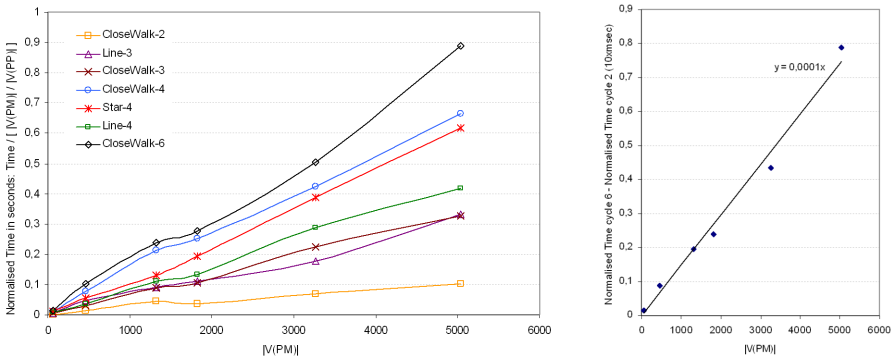


Fig. 5. Average response time of *u*EP-PMA algorithm on arbitrary random graphs for matching a star-like pattern, a line-like pattern and a pattern with a close-walk structure.

The Fig. 4-left side shows the average response time of *u*EP-PMA for the matching of three patterns with different structures and the same number of vertices on arbitrary random graphs. The line-like pattern requires less time in comparison with the star-like and close-walk patterns, providing an indication that the structure of the matched patterns influence the response time. The right side of the Fig. 4 shows the average response time of *u*EP-PMA on arbitrary random graphs for a same pattern and different number of vertices. The number of vertices of the pattern also influence the time response. In order to visualise the trend of the time response more clearly, we divided the time that the algorithm requires to compute a solution by the ratio between the number of vertices in the random graph (target graph) and the number of vertices in the pattern. The Fig. 5 - left side shows the trend of the normalised response time for all the different patterns considered in the experiment. The right side of the Fig. 5 illustrates the trend of the normalised time response for two patterns with different number of vertices. The trend lines in the two graphics of the Fig. 4 indicate that the time to solve the problem increase quadratically with the number of vertices on the target graph. The constant 6^{-7} suggest advantageous performance characteristics regarding the response time of the algorithm for small and medium size graphs.

6 Related work

Matching of process-centric descriptions is an activity in the context of service discovery and modelling of new services are activities. Patterns provide a notion of abstraction in models, and they can play a role in the reuse of previously implemented designs. A number of papers have proposed graph-based approaches for matching of processes and patterns. In [5] a technique for partial matches on behavioral models is presented. The proposal provides measures of semantic distance between resultant matches and user requirements. Several issues regarding complexity of the proposed algorithm are reported to be improved. However, the experimental results indicate a response time of approximately thirty seconds for a target graph of fifty vertices, which can be prohibitive for large processes. In [15] a method to measure distance between process definitions of

web services is presented. The method relies on a distance measure of normalised matrices representing graph-based process models. The proposed normalised matrices lack of flexibility in relation with chosen data structure. Optimisations on the computation of the normalised matrices, e.g. considering a more efficient data structure such as sparse matrices, is not mentioned. In [7] various types of structural matches for BPEL processes supporting dynamic binding of services are defined. BPEL processes are modelled as process trees where each tree node is an interaction. Activities which are not interactions are abstracted into internal steps and can not be matched. Duplicate interaction activities are not allowed in the tree. *Plugin* matching is presented as an approach based on a process simulation notion, however such as the authors indicate, the proposal requires further semantic analysis to decide if a process can replace other after a matching. In [6] the authors propose a measurement to compare two process models based on their observed behavior. Observed behavior is restricted to logs of process executions. Mining techniques are only applied over sequences of process steps rather than graphs representing process models. In [16] an best-effort method to exact and partial pattern matching is presented. The results of a matching process are presented to users ordered according to a proposed *goodness* measurement. The proposed method finds partial subgraphs in time linear on the size of the target graph. However, the pattern queries are limited in size and structure, and attributes or labels on edges are not considered. Neither overlaps nor hierarchical matching are considered.

7 Conclusion

In this paper we have discussed the benefits, the considerations and some possible solutions for a pattern-based approach for service identification. In its core, the approach uses a set of graph-based pattern matching algorithms. We discussed some considerations for exact, inexact, partial, generalised and hierarchical pattern matching. We provided a solution for exact and partial matching (*u*EP-PMA algorithm). We extend *u*EP-PMA by adding hierarchical pattern matching with the *u*H-PMA algorithm, and outlined a proposal for matching of generalised patterns (*u*G-PMA.). A solution for discovering frequent pattern in graphs (*u*EP-FPDA) was proposed. The solution attempts to discover utility patterns, which could provide a recommendation for designing new reusable technical-centered services.

Our initial motivation for this work was based on the potential benefits that pattern matching and pattern discovery techniques could provide to business analysts and architects during the definition of new services based on the analysis of process-centric models. Process models could be annotated with matches of process patterns and presented to the designers on standard modelling tools. This investigation assume the availability of process models or process-centric service descriptions and related patterns. The availability of process documentation - and with a unique type of process description- might be thought as quite difficult to find in real cases. However, we believe that business and architectural documentation in the form of process-centric models is becoming more and more relevant in the context of service architecture implementations. Models documenting real case scenarios are complex, numerous and often large. Thus, our proposal attempts to support designers by automating some of the steps during the analysis

and design activities of business process models and process-centric service architectures descriptions.

We believe that architecture abstractions, such as patterns, are a powerful concept that can be exploited to improve the design of new services and pattern matching techniques can help with the discovery of already implemented services. Further work regarding performance and scalability of our proposed algorithms is in development. We plan investigate their applicability to dynamic service composition.

References

1. Abramovsky, L., Griffith, R.: Outsourcing and offshoring of business services: How important is ICT? J. of the European Economic Association **4**(2-3) (2006) 594–601
2. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern-Oriented Software Architecture: On Patterns and Pattern Languages. Wiley and Sons (2007)
3. Erl, T.: Service-oriented architecture: Concepts, Technology, and Design. Prentice Hall (2004)
4. Wombacher, A., Rozie, M.: Evaluation of workflow similarity measures in service discovery. In Schoop, M., Huemer, C., Rebstock, M., Bichler, M., eds.: Service Oriented Electronic Commerce. Volume 80. GI (2006) 51–71
5. Corrales, J., Grigori, D., Bouzeghoub, M.: Bpel processes matchmaking for service discovery. In: On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. (2006) 237–254
6. Aalst, W.M.P.v.d., Medeiros, A.d., Weijters, A.: Process equivalence: Comparing two process models based on observed behavior. In: Business Process Management. (2006) 129–144
7. Eshuis, R., Grefen, P.: Structural matching of bpel processes. In: Fifth European Conference on Web Services. ECOWS07. (2007) 171–180
8. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, volume 2: Applications, Languages and Tools. World Scientific (1999)
9. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Information Systems **25**(2) (2000) 117–134
10. Aalst, W.M.P.v.d., Hofstede, A.H.M.t., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
11. Fettke, P., Loos, P.: Reference Modeling for Business Systems Analysis. IGI Publishing (2006)
12. Fiala, J.: Structure And Complexity of Locally Constrained Graph Homomorphisms. PhD thesis, Charles University, Faculty of Mathematics And Physics (2007)
13. Fiala, J., Kratochvíl, J.: Locally constrained graph homomorphisms–structure, complexity, and applications. Computer Science Review **2**(2) (2008) 97–111
14. Michihiro, K., George, K.: Finding frequent patterns in a large sparse graph*. Data Min. Knowl. Discov. **11**(3) (2005) 243–271
15. Bae, J., Liu, L., Caverlee, J., Rouse, W.B.: Process mining, discovery, and integration using distance measures. In: Proc. IEEE International Conference on Web Services (ICWS'06), IEEE Computer Society (2006) 479–488
16. Tong, H., Faloutsos, C., Gallagher, B., Eliassi-Rad, T.: Fast best-effort pattern matching in large attributed graphs. In: Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. KDD07, San Jose, California, USA, ACM (2007) 737–746
17. Hell, P., Nesetril, J.: Graphs and homomorphisms. Oxford Lecture Series in Mathematics and Its Applications **28** (2004)

8 Annex: Graphs

This annex is based on Nešetřil, Fiala and Hell's work [17],[13], [12].
A ***graph*** G is a set V_G of vertices together with a set E_G of edges, where each edge is a two-element set of vertices. If V_G is finite, the graph G is called a ***finite*** graph. If the graph has orientation, it is called ***directed*** graph, and each edge is called an *arc*. An arc can have one of the two orientations (u,v) or (v,u) with $u,v \in V_G$. If loops on vertices are allowed, then edges consist of only one vertex, written (u,u) with $u \in V_G$. A sequence of vertices of a graph G, such that the consecutive pairs are adjacent, is called a *walk* in G. If all vertices in a walk are distinct, then it is called a *path*. A graph G is called a ***connected*** graph if for every pair of vertices $u,v \in V_G$ there exists a finite path starting in u and ending in v. For a vertex u in a graph G, the set of all vertices adjacent to u are called the ***neighborhood*** of u and is denoted by $N_G(u)$, with $N_G(u) = \{v|(u,v) \in E_G\}$. Consequently, a vertex v is a neighbor of u if u and v are adjacent. A graph G is a ***subgraph*** of H if $V_G \subseteq V_H$ and $E_G \subseteq E_H$.

Homomorphisms. Graph homomorphisms are edge preserving vertex mapping between two graphs. A *graph homomorphism* from G to H denoted by $G \to H$ is a vertex mapping $f : V_G \to V_H$ satisfying $(f(u), f(v)) \in E_H$ for any edge $(u,v) \in E_G$. According to [13], whenever a homomorphism $G \to H$ is hold, then the image of the neighborhood of a vertex from the source graph V_G is contained in the neighborhood of the image of that vertex in the target graph V_H, i.e. $f(N_G(u)) \subseteq N_H(f(u))$ for all $u \in V_G$. Composition of two homomorphisms $f : F \to G$ and $g : G \to H$ is another homomorphism $g \circ f : F \to H$. If a homomorphism $f : G \to H$ is an one-to-one mapping and f^{-1} is also a homomorphism, then f is called an ***isomorphism***. In such a case is said that G and H are isomorphic and it is denoted by $G \simeq H$. An isomorphism $f : G \to G$ is called an automorphism of G, and the set of all automorphisms of G is denoted by $AUT(G)$. Using the latter notation, for graphs G and H three kind of homomorphic mapping are defined as:
- $G \xrightarrow{B} H$ if there exist a *locally bijective homomorphism* $f : V_G \to V_H$ that satisfies for all $u \in V_G : u \in V_G : f(N_G(u)) = N_H(f(u))$ and $|f(N_G(u))| = |N_G(u)|$.
- $G \xrightarrow{I} H$ if there exist a *locally injective homomorphism* $f : V_G \to V_H$ that satisfies for all $u \in V_G : |f(N_G(u))| = |N_G(u)|$.
- $G \xrightarrow{S} H$ if there exist a *locally surjective homomorphism* $f : V_G \to V_H$ that satisfies for all $u \in V_G : f(N_G(u)) = N_H(f(u))$.

Note that for the mappings above, locally bijective homomorphism is both locally injective and surjective. The mappings are also known in the literature as (full) covering projections (bijective), or as partial covering projections (injective), or as role assignments (surjective). Additionally, any locally surjective homomorphism f from a graph G to a connected graph H is globally surjective, and any locally injective homomorphism f from a connected graph G to a forest H is globally injective [12].

Labelled Graphs. The graph $G = (V_G, E_G, \ell_{V_G}, \ell_{E_G})$ is a graph where the vertices in V_G and edges in E_G have labels. The functions assigning labels to vertices and edges are surjective homomorphisms $\ell_{V_G} : V_G \to L_{V_G}$ and $\ell_{E_G} : E_G \to L_{E_G}$ for all the vertices in V_G and the edges in E_G, respectively. L_{V_G} and L_{E_G} are the sets of vertex labels and edge labels, respectively. Note that surjection allow the existence of a same label in $L_{V_G}(L_{E_G})$ for several vertices(edges).

Server-side Exception Handling by Composite Web Services

Kung-Kiu Lau and Cuong Tran

School of Computer Science, the University of Manchester
Manchester M13 9PL, United Kingdom
kung-kiu,ctran@cs.man.ac.uk

Abstract. Currently exception handling for web service orchestrations is performed on the client side. We have defined composite web services [11] that are not single orchestrations but complete web services that contain all possible orchestrations of their sub-services. Our composite web services can therefore define and perform exception handling just once for all such orchestrations, on the server side. In this paper we explain and discuss our approach to server-side exception handling by composite services.

1 Introduction

Currently in web services, client applications are orchestrations of web services provided by various web servers, and exception handling for these applications is defined and performed on the client side (Fig. 1). To be more precise, exception handling is

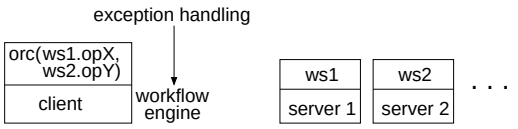


Fig. 1. Client-side exception handling for an orchestration.

performed by the workflow engine on the client side during its execution of an orchestration of web services, e.g. *orc*(*ws1.opX,ws2.opY*) in Fig. 1.

In [11] we defined composite web services that are not orchestrations. An orchestration defines just one workflow for invoking a fixed set of operations, e.g. *opX* in *ws1* and *opY* in *ws2* in *orc*(*ws1.opX,ws2.opY*) in Fig. 1. In contrast, our composite service is a web service offering operations that can invoke all the operations in all its sub-services. In other words, our composite service contains all possible orchestrations of its sub-services.

It follows that our composite service should be able to define exception handling just once for all possible orchestrations of its sub-services. This is clearly an advantage, compared to defining exception handling for one possible orchestration at a time (e.g. *orc*(*ws1.opX,ws2.opY*) in Fig. 1). Moreover, since our composite service is implemented on a server, its exception handling is now performed on the server side (Fig. 2). The benefit of server-side exception handling is that client applications using any or-

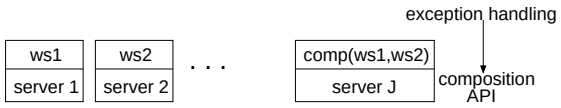


Fig. 2. Server-side exception handling by a composite service.

chestrations contained in the composite service (e.g. *comp*(*ws1,ws2*) in Fig. 2) need not define and perform the exception handling that the composite is already providing. Moreover, it is possible to include recovery actions in a composite service's exception handling, and by so doing, we can make the composite service more stable and reliable from the point of view of all its clients.

In this paper, we show how we define and perform exception handling in composite services on the server side. It is worth noting that our work does not mean to handling all sorts of exception as we shall say clearly in the following sections but to propose a distinct and significant approach to handle exceptions.

There are many exceptions that can be found in web service composition [14]. In our work, we focus on the infrastructure exception *Unavailability*, the process-defined exception *Timeout*, the application exception *Fault*, and suitable recovery actions for these exceptions. There are no obvious, sensible recovery actions for the other exceptions. If the workflow management system fails, there is nothing much we can do about the resulting *Failure* exception. Similarly it is not clear what recovery action is appropriate when the *Delay* or *QoS* exception occurs.

2 Composite Web Services

In our previous work [11], we have defined composite web services. In this section, we give only a brief account of these services. For more details of our composite web services, we would like to advise readers to refer to [11].

We define composite services as distinct from orchestrations. A composite web service *comp*(*ws1,ws2,*. . .) is a composition of web services (not just their operations, as in orchestrations), where *comp* is a function with the type $comp : ws \times ws \times \cdots \times ws \rightarrow ws$, where *ws* is the type of web services. The composite *comp*(*ws1,ws2,* . . .) is thus a whole web service.

This kind of composition is different from an orchestration *orc*(*ws1.opX,ws2.opY,* . . .), which defines a workflow for invoking operations in web services. The function *orc* has the type $orc : op \times op \cdots \times op \rightarrow wf$, where *op* is the type of operations in web services, and *wf* is the type of workflows for invoking a set of such operations.

Our composite service thus composes whole services into another whole service. It does so by using composition operators defined in our component model [12, 10].

In our model, components have the distinguishing features of *encapsulation* and *compositionality*. Components are constructed from two kinds of basic entities: (i) *computation units*, and (ii) *connectors* (Fig. 3). A computation unit *CU* encapsulates computation. It provides a set of methods (or operations). *Encapsulation* means that *CU*'s methods do not call methods in other computation units; rather, when invoked, all their computation occurs in *CU*. Thus *CU* could be thought of as a web service.

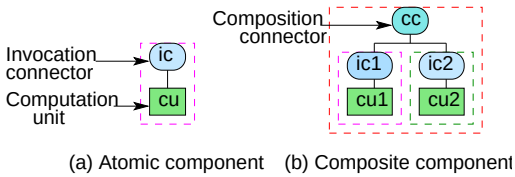


Fig. 3. Our component model.

There are two kinds of connectors: (i) *invocation*, and (ii) *composition* (Fig. 3). An invocation connector is connected to a computation unit *CU* so as to provide access to the methods of *CU*.

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

Clearly composition connectors can define (and encapsulate) *workflow* for a set of connected components. They can define workflow control-flow for sequencing, branching and looping, as described in e.g. [16].

Components are defined in terms of computation units and connectors. There are two kinds of components: (i) *atomic*, and (ii) *composite* (Fig. 3). An atomic component consists of a computation unit with an invocation connector that provides an interface to the component. An atomic component encapsulates *computation*. A composite component consists of a set of components (atomic or composite) composed by a composition connector. The composition connector provides an interface to the composite. A composite component encapsulates *computation* and *control*.[1]

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

Our components are also *compositional*, i.e. the composition of two components C_1 and C_2 yields another component C_3. In particular, $C3$ also has the defining characteristics of encapsulation and compositionality. Encapsulation and compositionality lead to *self-similarity* of composite components, i.e. a composite has the same structure as any of its sub-components, as can be clearly seen in Fig. 3(b). Self-similarity provides the basis for a hierarchical way of composing systems from components.

We used our model as a component model for web services. We can use standard web services as atomic components, composite web services as composite components, and use the composition connectors as composition operators for web services. This is

[1] We do not consider data encapsulation [15] here, for simplicity.

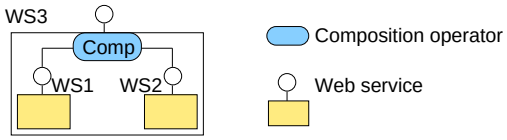


Fig. 4. Composite web services.

illustrated in Fig. 4, where two standard web services *WS1* and *WS2* are composed by a composition operator *Comp* into a composite service *WS3*.

WS3 is a web service, just like *WS1* and *WS2*. However, whereas *WS1* and *WS2* have interfaces described in standard WSDL, *WS3* has an interface that cannot be described in standard WSDL, because *WS3* contains workflow embodied in the composition operator *Comp*. For instance, a *pipe* connector would introduce a workflow structure that connects a number of services, and sequentially invokes each service and uses the result as input to the next invocation; a *selector* connector would provide a branching structure for choosing a service from a set of services, according to a branching or choice condition.

Therefore, in order to define *WS3* as a web service, we need to extend standard WSDL so as to incorporate workflow description. Then we need to devise a method to generate its interface in the extended WSDL from the standard WSDL interfaces of *WS1* and *WS2*. Accordingly, in [11] we defined an extended form of WSDL that contains new <*portType*> elements called <*workflow*>, <*pipe*> and <*choice*> for defining the workflow in a composite service, in a pipe connector and in a selector connector respectively. We also implemented a method for generating the interface of a composite service in extended WSDL from the WSDL interfaces of the composed standard web services.

Example 1. Consider a bank system with just one ATM that serves two bank consortia $B1$ and $B2$, each with two bank branches, $BB1$ and $BB2$, $BB3$ and $BB4$ respectively. The ATM reads the customer's card, performs a security check, identifies the customer's bank consortium and passes customer requests together with customer details to the customer's bank consortium. The customer's bank consortium checks customer details and identifies the customer's bank branch, and then passes on the customer requests and customer details to the customer's bank branch. The bank branch checks customer details and provides the usual services of withdrawal, deposit, balance check, etc.

The bank system can be built as a composite web service composed from standard web services for *ATM*, *B1, B2, BB1, BB2, BB3* and *BB4* (Fig. 5). $P1$, $P2$ and $P3$ are

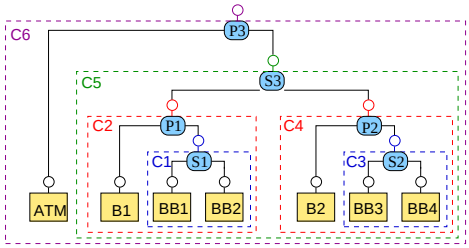


Fig. 5. The bank composite web service.

pipe composition connectors; and $S1$, $S2$ and $S3$ are selector composition connectors. The top-level connector $P3$ is the interface to the system, and is where control flow starts.

The composition is hierarchical (composite services are denoted by dotted boxes): *BB1* and *BB2* are composed into the composite service *C1* by using the selection connector *S1*; the composite *C1* is in turn composed with *B1* using the pipe connector *P1*, creating the composite *C2*; similarly *BB3* and *BB4* are composed into *C3* by using the selection connector *S2*; the composite C3 is then composed with *B2* using the pipe connector *P2*, creating the composite *C4*; the composite *C2* is in turn composed with *C4* by using the selector connector *S3* to create the composite *C5*; the composite *C5* is composed with *ATM* by using another pipe connector *P3*, creating the composite *C6*.

The composite service C6 provides all the operations offered by its sub-services. The workflow of the composite service C6 is outlined as follows:

```
<workflow>
 <pipe> <set name="ATM">
   <operation name="gbc">...</operation> </set>
  <choice>
   <case condition="1">
    <pipe>
     <set name="B1"> <operation name="gbr">...</operation> </set>
     <choice>
      <case condition="1"><set name="BB1">
       <operation name="wd">...</operation>
       <operation name="dp">...</operation> ...</set> </case>
      <case condition="2"><set name="BB2">
        <operation name="wd">...</operation>
        <operation name="dp">...</operation> ...</set></case>
     </choice></pipe></case>
   <case condition="2">
    <pipe>
     <set name="B2"> <operation name="gbr">...</operation> </set>
     <choice>
      <case condition="1"><set name="BB3">
  ...
```

This workflow first invokes the 'get bank consortium' (gbc) operation of ATM, and pipes the result to the branching structure; if the result is 1, then the 'get bank branch' (gbr) operation of B1 is invoked, or if the result is 2 then the 'get bank branch' operation of B2 is invoked. The result of B1's operation is compared with the branching condition; if the value is 1, then *any* one of BB1's operations ('withdrawal' (wd) or 'deposit' (dp)) can be invoked, or if the value is 2, then *any* one of BB2's operations (wd or dp) can be invoked. Similarly the result of B2's operations will lead to the invocation of operations of BB3 and BB4 (not shown here). After that, the workflow ends and the result of the last invocation is returned.

In [11], we gave a detailed implementation of composite web services, as outlined in this section. This implementation provides the basis for our work in this paper.

3 Exception Handling by Composite Services

Our definition of composite services makes it possible to add exception handling to composite services in a hierarchical manner.

In this section, we describe how we can make a composite web service handle common exceptions, as well as provide sensible recovery actions, while invoking its sub-services, which in turn invoke their sub-services, and so on. At each level of sub-services, we shall deal with the following exceptions: *Unavailability*, *Timeout*, *Fault* (see [14]); and we shall provide the following recovery actions: *Retry* and (implicit) *Replace*.

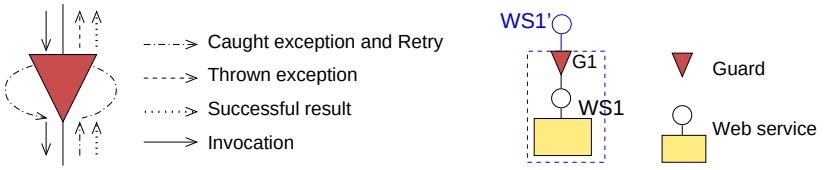
Retry will redo the invocation of the sub-service up to n times. If there is a successful response within n times, it is passed back to the caller service. Otherwise, an exception is raised and returned to the caller. The *Retry* action has a default minimum and maximum value for n. Without a valid n value given by the caller, this default value is used.

Replace and retry will invoke an alternative service to replace the service that has failed. If there is a successful response from the replacement service, it is passed to the caller service. Otherwise, continue to retry on another alternatives until *Retry* reaches the boundary value n. Note that, alternative services are chosen by the composite service (implicitly) without caller intervention.

To implement exception handling with these recovery actions, we again use our component model, and integrate the exception handling mechanism into our component model semantics. More precisely, composition connectors (i.e. pipe, selector and sequencer) will intercept all exceptions raised during the invocation of sub-service operations and simply throw them back as normal returns. The whole complex exception handling mechanism will be defined in two new entities added to our component model: *exception guard* and *exception facade*.

3.1 Exception Guard

An *exception guard* is a unary connector that is connected to the interface of a (sub)service (Fig. 6). It receives invocation requests to provided operations in the service and



(a) Exception guard (b) Service with exception guard

Fig. 6. Exception guard.

intercepts all the results. It captures any exceptions raised by invocations to the service. For exceptions that are *Unavailability*, *Timeout* and *Fault*, it performs the *Retry* recovery action. For other exceptions, it will simply throw them.

An exception guard thus acts as a filter that provides exception handling together with recovery actions. Applying a guard to a service results in a new service, e.g. *WS1* with guard $G1$ becomes *WS1'* in Fig. 6(b). The new service encapsulates the original service and exposes its interface through an extended WSDL document, as we shall see later, to express exception handling semantics. In order to allow the caller a choice of exception handling options, for every provided operation of the original service, we

generate two provided operations for the new service: (i) one operation that invokes the original operation and handles *Unavailability* and *Timeout* exceptions with the *Retry* recovery action; (ii) one operation that invokes the original operation and simply throws any exception encountered. The default and valid range of timeout and retry values used for handling exceptions are preset properties of the exception guard, and are defined in extended WSDL for the new service.

Thus all the provided operations of the original service are made available through the interface of the new service, together with additional exception handling semantics defined by the guard.

For example, in Fig. 6(b), for simplicity, let us assume that *WS1* provides just one operation *op1* that takes one input and returns one output, both as strings. Thus its signature is:

```
op1 (String param) : String
```

The new service *WS1'* provides two operations with exception handling. Their signatures are as follows:

```
op1_RT (String param, String timeout, String num_of_retries) : String
op1_O (String param) : String
```

op1_RT has three inputs: the first one is the input for computation; the second is the value of timeout, and the third one is the number of retries in case of invocation failure. *op1_O* has only one input, which is the input for computation, and whatever exceptions that result will not be handled but simply thrown back.

3.2 Exception Facade

An *exception facade* is an n-ary connector that connects a number of services and produces a new composite service (Fig. 7). The idea behind an exception facade is to unify

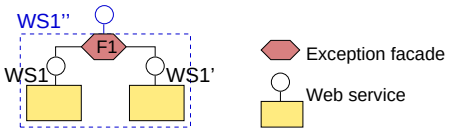


Fig. 7. Exception facade.

two or more services that provide the same operations in order to increase the reliability of operation invocations. If one service cannot respond to a call to one of its operations, then an alternative service is used to replace the failed service. The services are prioritised according to their rankings as backup services, and there is a preset bound on the number of *Retry* actions for the composite service in case of exceptions.

The exception handling behaviour is defined as follows. When there is an invocation to a provided operation, an exception facade delegates the call to the sub-service with the highest priority and intercepts any results. In case of success, the facade passes the result back to the client. In case of exception, the facade recovers by *Retry* actions, provided the bound on retries has not been exceeded.

In detail, if the exception is *Unavailability* or *Timeout*, the facade retries to delegate the invocation to the sub-service with the next highest priority. When there is no more

sub-service with a lower priority, and the number of retries has not reached the bound, the facade starts from the sub-service with the highest priority again. When the number of retries approaches the limit and exception persists, the exception is thrown to the caller. Other exceptions are intercepted but simply thrown back.

As in the case for an exception guard, the default and valid range of timeout and retry values used for handling exceptions are preset properties of an exception facade.

An exception facade thus unifies sub-services and provides an interface for the new service. In order to allow the caller a choice of exception handling options, for each duplicate set of (semantically equivalent) operations provided by the sub-services, two operations are generated for the new service. The interface of the new service will be defined in extended WSDL to expose the generated operations and properties.

For example, in Fig. 7, an exception facade *F1* is applied to two sub-services, *WS1* and *WS1'*, to produce new service *WS1"*. In this example, *WS1'* can be imagined as a backup service for the main service *WS1*, and so their interfaces are identical. *WS1'* is the new service that is reliable and convenient to use, as it provides exception handling for every provided operation.

Again for simplicity, let us assume that the two sub-services *WS1* and *WS1'* have just one operation. The operation accepts one input as a string and return an output also as a string. Thus its signature is:

```
op1 (String param) : String
```

The new service *WS1"* therefore has two operations that have the following signatures:

```
op1_RT (String param, String timeout, String num_of_retries) : String
op1_O (String param) : String
```

op1_RT has three inputs: the first one is the input for computation; the second is the value of timeout and the third is the number of retries in case of invocation failures. *op1_O* has only one input, which is input for computation and whatever exceptions result will not be handled and simply thrown back.

4 Defining Composite Web Services with Exception Handling

A standard web service has its interface described in a WSDL document. Our composite web service has an interface in an extended form of WSDL [11]. Now with exception handling, we are introducing yet more new semantics to composite web services. Therefore, the interface of a composite web service should be further extended with a new element to capture such semantics.

We add one new element, namely <*exception*>, into the conventional <*portType*> element. The <*exception*> element consists of two child elements, <*timeout*> and <*retry*>. Each of these elements has attributes such as *defval*, *minval* and *maxval* to represent default, minimum and maximum value of timeout and retries. The syntax of the extended WSDL is as shown in Fig. 8. For example, the service *WS1'* in Fig. 6 which is built from applying a guard to the service *WS1* will have the extended WSDL interface shown in Fig. 9. The original service *WS1* provides one operation, namely *op1*. The new service *WS1'* has two operations, *op1_O* and *op1_TR*. The *op1_O* operation is identical to the *op1* operation provided by *WS1* while *op1_TR* has an exception

```
<wsdl:portType name="nmtoken">*    <wsdl:documentation .... />?
  <wsdl:operation name="nmtoken">*    <wsdl:documentation .... />?
    <wsdl:input name="nmtoken"? message="qname">?    <wsdl:documentation .... />?   </wsdl:input>
    <wsdl:output name="nmtoken"? message="qname">?    <wsdl:documentation .... />?   </wsdl:output>
    <wsdl:fault name="nmtoken"? message="qname">?    <wsdl:documentation .... />?   </wsdl:fault>
  </wsdl:operation>
  <exception>?
    <timeout defval="nmtoken" minval="nmtoken" maxval="nmtoken">?
    <retry defval="nmtoken" minval="nmtoken" maxval="nmtoken">?
  </exception>
</wsdl:portType>
```

Fig. 8. Extended WSDL for exception handling.

```
<wsdl:portType name="W1'">*
  <wsdl:operation name="op1_O">
    <wsdl:input name="param" message="inMessage">   </wsdl:input>
    <wsdl:output name="return" message="outMessage">    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="op1_TR">
    <wsdl:input name="param" message="inMessageTR">   </wsdl:input>
    <wsdl:output name="return" message="outMessageTR">    </wsdl:output>
    <wsdl:fault name="exception" message="faultMessageTR">    </wsdl:fault>
  </wsdl:operation>
  <exception>
    <timeout defval="−1" minval="100" maxval="30000">
    <retry defval="0" minval="0" maxval="5">
  </exception>
</wsdl:portType>
```

Fig. 9. Extended WSDL interface of *WS1'*.

handling mechanism. Invoking *op1_TR* allows us to use exception handling through the parameters of its invocation.

The extended WSDL interface is connected to a composition connector, e.g. a pipe or a selector, in any composition to build further composite services. The interface of such a composite service therefore has a workflow structure (introduced by the pipe and selector connector) as well as exception handling semantics.

Example 2. Consider the bank system in Example 1, as a composite web service (Fig. 5). All the sub-services may be located on different networks; that is, the bank system may be spread over different networks and different platforms. This is where problems arise. In general the bank system can encounter hardware and software problems such as server hardware failure, server software failure, network disconnection or congestion, etc.

For example, if the connection to bank branch service *BB1* is broken, then customers having accounts in this bank branch cannot use the bank services. These problems which are normally unexpected can cause the whole system to malfunction or totally broken down. Thus, this problem must be addressed in order to build a secure and reliable system.

We can define the bank system as a composite service with exception handling, by adding exception guards and facades. The result is depicted in Fig. 10. We assume that all the bank branch services, BB1, BB2, BB3 and BB4 have backup services, BB1', BB2', BB3' and BB4' respectively; services B1, B2 and ATM do not have backup services. To start building the composite bank service, every pair of main service and backup service is first composed with facades F1, F2, F3 and F4 to achieve four reliable

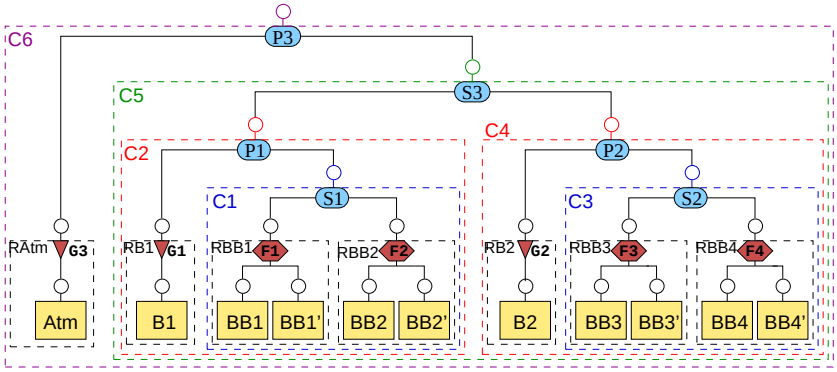


Fig. 10. The bank composite web service with exception handling.

bank branch services RBB1, RBB2, RBB3 and RBB4 respectively. They will then be composed by two selector connectors S1 and S2 to achieve two composite services C1 and C2. B1 and B2 do not have backup service so we apply two exception guards G1 and G2 to them to make two reliable bank services RB1 and RB2, which will be composed with two previously built composite service C1 and C2 respectively, by using two pipe connectors P1 and P2 to achieve two bigger composite services C3 and C4 respectively. We then compose C3 and C4 by using selector S3 to get composite C5. The ATM service also does not have a backup service, so we apply the exception guard G3 to make reliable service RATM, and then compose it with the composite C5 using the pipe connector P3, to get the composite C6. C6 represents the composite bank service with exception handling by providing all services with exception handling.

The interface of the composite C6 is described in extended WSDL, and is outlined as follows:

```
<workflow>
 <pipe> <set name="RATM"> <operation name="getbank_O">...</operation>
                          <operation name="getbank_TR">...</operation>
        <exception> <timeout defval="-1" minval="100" maxval="30000"/>
                    <retry defval="0" minval="1" maxval="5"/> </exception> </set>
  <choice>
   <case condition="1">
    <pipe> <set name="RB1"> <operation name="getbranch_O">...</operation>
                            <operation name="getbranch_TR">...</operation>
           <exception> <timeout defval="-1" minval="500" maxval="30000"/>
                       <retry defval="0" minval="1" maxval="5"/> </exception> </set>
     <choice>
      <case condition="1"><set name="RBB1">
       <operation name="withdraw_O">...</operation>
       <operation name="withdraw_TR">...</operation>
       <operation name="deposit_O">...</operation>
       <operation name="deposit_TR">...</operation> ...
       <exception> <timeout defval="-1" minval="500" maxval="10000"/>
                <retry defval="0" minval="0" maxval="5"/> </exception> </set> </case>
      <case condition="2"><set name="RBB2">
         ...
       <exception>
        <timeout defval="-1" minval="1000" maxval="10000"/>
        <retry defval="0" minval="2" maxval="10"/> </exception>
      </set></case></choice></pipe></case>
   <case condition="2">
    <pipe>
     <set name="RB2">
  ...
```

This workflow first invokes *any* operation of RATM, and pipes the result to the branching structure; if the result is 1, then *any* one of RB1's operations can be invoked, or if the result is 2 then *any* one of RB2's operations can be invoked; the result of RB1's operation is used to compare with the branching condition; if the value is 1, then *any* one of RBB1's operations can be invoked, or if the value is 2, then *any* one of BB2's operations can be invoked. Similarly the result of RB2's operations is used in checking the branching condition; if the condition is 1 then *any* one of RBB3's operations or RBB4's operations will be invoked. After that, the workflow ends and the result of the last invocation is returned. The integrated exception handling mechanism can be used in every invocation of any operation provided by any sub-service in this workflow by selecting the appropriate operation according to its signature, e.g. an operation name ending in *_TR*.

From the point of view of its clients, C6 appears reliable because it handles exceptions with recovery actions. Clients need not repeat the exception handling already performed in C6. They can simply invoke C6 with operations and exception handling mechanisms as parameters, as specified in its interface.

5 Implementation

To implement composite services with exception handling as defined in the previous sections, we make use of our existing implementation of composition connectors (i.e pipe and selector), and extend our existing implementation of composite services by adding the implementation of the exception guard and exception facade connectors. Our existing implementation [11] is in Java and the Axis framework [3].

Applying a guard connector to one service or applying a facade connector to two services results in a (composite) service with an interface in extended WSDL. We need to generate a Java implementation for such an interface. The implementation will dispatch any requests for operations in the interface of the composite to the implementation of the guard or facade connector, and pass any result from the latter back to the client.

Our implementation of the exception guard and the exception facade connector has two (overloaded) operations both called *invoke*. For simplicity, our implementation only deals with parameters of primitive data types, e.g. string, integer, float, etc. We use String as intermediate type because other primitive types can be converted to String and vice versa. In addition, the implementation detects exceptions, e.g. *Unavailability*, *Timeout* and so on, as shown in these signatures:

```
String invoke (String oper, String[] params)
     throws UnavailableException, TimeoutException, Exception

String invoke (String oper, String[] params, int timeout, int retry)
     throws UnavailableException, TimeoutException, Exception
```

Thus the exception guard or the exception facade connector receives an operation, a list of parameters, together with timeout and retry values optionally depending on which operation receives the request. If the first *invoke* operation receives the request, it is invoked with the given list of parameters, and a successful result or any exception will be returned as an output message or a fault message respectively by the composite service. If the second *invoke* operation receives the request, it is invoked with the list of

parameters, and it sets the timeout of this invocation according to the timeout value. A successful result is returned as a normal result. If there is a *Timeout* or *Unavailability* exception, the connector will recover by retrying the invocation to the appropriate service (the same service in the case of a guard; an alternative service in the case of facade). If the exception still persists, it will be returned as a fault message. If the exception is of other types, it will also be returned as a fault message by the composite service.

Although it is expressed in a form of WSDL which is further extended from that in our previous work, the interface of a composite with a guard or facade connector has a top-level <*workflow*> element that is the same as before. As a result, we can use our existing composition connectors in [11], i.e. pipe and selector, on these composites to build bigger composite services. These connector will therefore expose operations provided by sub-services through interfaces to client as already defined in our previous work [11].

Briefly, the Java class of a pipe or selector connector always has one operation *invoke*, that is the operation provided by the composite service to the outside world. Clients use a composite service via its *invoke* operation.

Basically, the signature of *invoke* comprises three main elements, viz. condition, operation names and operation parameters, plus return and exception types. The condition is used in a branching workflow structure for selecting sub-services. Other elements have the same meaning as in the exception guard and exception facade connector.

Each operation is provided by a sub-service which could be either a service with exception handling, or a composite service. If a sub-service is service with exception handling, the connector identifies the number of parameters for every operation from its interface so that the connector can call it with the correct parameters extracted from the parameter list. The result that is either a success or an exception is returned as the output of the composite. If a sub-service is a composite service, the connector just passes the whole operation list at that point to the *invoke* operation of the sub-service.

Finally, in order to demonstrate how our exception handling approach performs in practice, we created a random test case where we simulate *Unavailability* and *Timeout* exceptions. The data for the test case is shown in Table 1, namely services, response times and hosting servers. All the reliable sub-services, RAtm, RB1, RB2, RBB1, RBB2, RBB3, RBB4, and composite services C1, C2, C3, C4, C5, C6 are created and deployed onto one reliable server, because we do not have more servers available, but we can easily extend our test case to many such servers.

Service	Response time	Location
ATM	No delay	Server 1
B1, BB1	Delay 0-1sec	Server 1
BB2	Service moved	Server 1
BB1', BB2'	Delay 0-2sec	Server 2
B2	Delay 0-2sec	Server 4
BB3, BB4	Server down	Server 3
BB3'	Delay 0-1sec	Server 4
BB4'	Delay 0-1.5sec	Server 4
RAtm, RB1, RB2, RBB1, RBB2, RBB3 and RBB4	-	Reliable server
C1, C2, C3, C4, C5 and C6	-	Reliable server

Table 1. Test case data.

Fig. 11 shows the composite bank service handling a balance checking request and the resulting exceptions step by step. When the composite bank service C6 receives a

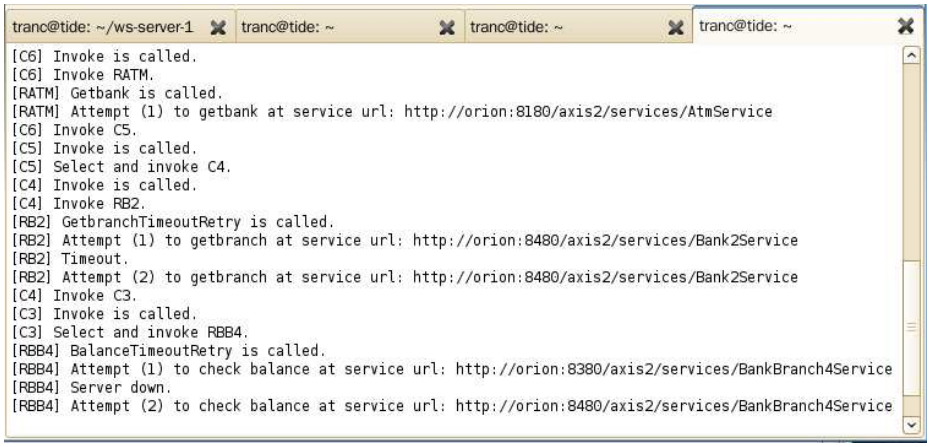


Fig. 11. Test case result.

request, it invokes the get bank operation of sub-service RATM. The result is that bank id is piped to composite service C5. C5 then uses the value to select service C4 which invokes the get branch operation of service RB2. RB2 encounters a timeout exception, and retries to call its sub-service again. The second call of RB2 is successful, and the result is that the branch id is passed to service C3. C3 uses the branch id to select branch service RBB4 to invoke. Again, another exception is encountered and is handled by retrying invocation of another service located at different server.

6 Discussion and Related Work

Our approach offers server-side exception handling by composite web services of some common exceptions. In the literature we have not found an equivalent approach to ours.

The main advantage of our approach is that we define and perform exception handling once, for all the orchestrations contained in a composite service. Existing approaches do not use composite services, but use individual orchestrations, and as a result they have to define exception handling for every orchestration.

For example, let us use the bank example to compare our approach with the *try-catch* approach supported by BPEL for client-side exception handling approach. Suppose there are two clients who want to build their own orchestrations from given atomic (standard) services introduced in our example. One orchestration provides the *withdraw* service and can handle two exceptions, *Unavailability* and *InvalidWithdraw*. The other orchestration provides the *balance checking* service and handles one exception, *Unavailability*. In BPEL, both composition and exception handling are defined at the level of operations. The workflow only composes several specific operations and provides

exception handling for them. Different clients who wish to compose different specific operations of the same services, need to define different workflows together with exception handling mechanisms. Thus, to build the above two applications, each client using client-side exception handling needs to build a workflow with its own exception handling mechanism.

Another advantage of our approach is that a composite service with exception handling is a reliable service, from the point of view of all its clients, whatever orchestration (contained in the composite service) they are using.

So far, all the approaches to exception handling we have found are for the client side. We summarise them here.

[13, 5, 6] present some taxonomies of exceptional situations. [5] also discusses exception handling in workflow management systems. At the lowest level, BPEL [2] provides the primitive try-catch construct to specify the exception to be caught, and define compensation actions as part of workflow specification.

In [8], a processor is built to inject codes (try-catch structures) into a BPEL orchestration, so that the resulting BPEL workflow when encountering exceptions will request for alternative services by invoking an external service.

The approaches in [4] and [13] use the ECA rule for modelling exceptions. The ECA rule in [4] is stored and used by a processor which interacts with the commercial FORO workflow engine to handle exception. In [13], the JECA rule is processed and combined with a CBR (Case-Based Reasoning) mechanism to enhance exception handling. Similarly, ADOME-WFMS [7] also uses ECA to model exception and resolution which is integrated in the ADOME workflow management system.

In [1], an extension to the YAWL workflow engine is created to allow defining exception handling rule sets and associated recovery actions (defined by Exlet) which will be consumed a workflow engine.

In [14, 9], a policy is used to specify what exception can be captured and how exception can be resolved. The policy is either used to generate exception handling construct for the target BPEL workflow [14] or processed by a workflow middleware [9].

7 Conclusion

In this paper we propose an approach to server-side exception handling by composite web services. Currently, our composite services can capture *Unavailability* and *Timeout* exceptions and provide *Retry* as recovery action, or *Throw* to propagate exceptions. Since our composite service contains all possible orchestrations of its sub-services, clients can use any such orchestration without needing to repeat the exception handling already provided by the composite service. As a result, our composite service is reliable, from the point of view of all its clients. Thus our approach offers important benefits to clients for building service oriented applications.

In future work, we intend to investigate sensible resolution for other exceptions such as *Delay* and *QoS Degradation*. This will add even more value to our server-side exception handling approach.

References

1. W. M. P. van der Aalst *et al*. Dynamic and extensible exception handling for workflows: A service-oriented implementation. Technical report, BPM Center, March 2007.
2. T. Andrews *et al*. BPEL4WS - version 1.1. Technical report, IBM, 2003.
3. Apache. Axis - web services framework web site. `http://ws.apache.org/axis2/`.
4. F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi. Specification and implementation of exceptions in workflow management systems. In *TODS*, volume 24, pages 405–451, 1999.
5. F. Casati and G. Cugola. Error handling in process support systems. In A. Romanovsly et al., editor, *Exception Handling*, pages 251–270. Springer-Verlag Berlin Heidelberg, 2001.
6. K.S. May Chan, J. Bishop, J. Steyn, L. Baresi, and S. Guinea. A fault taxonomy for web service composition. In *Proceedings of WESOA07*. Springer LNCS, 2007.
7. D. K. W. Chiu *et al*. Adome-wfms: Towards cooperative handling of workflow exceptions. In A. Romanovsky et al., editor, *Exception Handling*, pages 271–288. Springer, 2001.
8. K. Christos, V. Costas, and G. Panayiotis. Enhancing bpel scenarios with dynamic relevance-based exception handling. In *ICWS*, pages 751–758, 2007.
9. A. Erradi, P. Maheshwari, and V. Tosic. Recovery policies for enhancing web services reliability. In *IEEE International Conference on Web Services (ICWS'06)*, 2006.
10. K.-K. Lau, L. Ling, and Z. Wang. Composing components in design phase using exogenous connectors. In *Proceedings of 32nd ECSEAA*, pages 12–19, 2006.
11. K.-K. Lau and C.M. Tran. Composite web services. In *In C. Pautasso and T. Gschwind, editors,* Proceedings of 2nd Workshop on Emerging Web Services Technology, 2007.
12. K.-K. Lau *et al*. Exogenous connectors for software components. In G. Heineman *et al.*, editor, Proc. 8th Int. Symp. on CBSE, LNCS 3489. Springer, 2005.
13. Z. Luo, A. Sheth, K. Kochut, and J. Miller. Exception handling in workflow systems. In *Applied Intelligence*, pages 125–147. Kluwer Academic, 2000.
14. L. Zeng, H. Lei, and Boualem Benatallah. Policy-driven exception-management for composite web services. In *Proceedings of CEC05*. IEEE, 2005.
15. K.-K. Lau and F. Taweel. Data encapsulation in software components. In *In H.W. Schmidt* et al.*, editor,* Proc. 10th Int. Symp. on Component-based Software Engineering, LNCS 4608, pages 1–16. Springer-Verlag, 2007.
16. W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. In Distributed and Parallel Databases, pages 5–51, 2003.

Towards Flexible Interface Mediation for Dynamic Service Invocations

Philipp Leitner, Anton Michlmayr, Schahram Dustdar

Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8/184-1
A-1040, Vienna, Austria
lastname@infosys.tuwien.ac.at

Abstract. One of the main benefits of service-based systems is the loose coupling of components, which increases flexibility during the selection of internal and external business partners. However, currently this flexibility is severely limited by the fact that components have to provide not only the same functionality, but do so via virtually the same interface. Invocation-level mediation may be used to overcome this issue – by using mediation interface differences can be resolved transparently at runtime. In this paper we present the general concepts of invocation-level mediation, and show how these ideas are integrated into our dynamic service invocation framework Daios. To demonstrate the flexibility of our mediation framework we present two fundamentally different mediation strategies, one based on structural similarity and one based on semantically annotated WSDL.

1 Introduction

Systems based on the Service-Oriented Architecture (SOA) paradigm [1] decouple clients from service providers by leveraging standardized protocols and languages (e.g., HTTP, SOAP, WSDL) and a registry (e.g., UDDI or the ebXML registry) as service broker. In theory, this loose coupling allows service clients to roam freely between internal and external business partners, and always select the partner that is most appropriate at any given time. However, in practice this flexibility is limited by the problem that clients rely on specific service interfaces for their invocation. Therefore, services need to adhere to identical WSDL contracts in order to be interchangeable at runtime. The assumption of interface compatibility is not realistic if services are provided by different departments or companies.

Currently, most work in the area focuses on providing an infrastructure to resolve these compatibility issues: ESBs [2] provide an additional bus that decouples clients and services, and integration adapters or mediators [3, 4] are used as intermediary to resolve the inherent problems of invocation heterogeneity. The approach that we present in this paper follows a different idea: we use a pure client-side approach to mediation, i.e., we enable the clients themselves to

adapt their invocation to different target services. Specific mediation behavior is introduced in the clients using mediation adapters, which can either be general-purpose or tailored towards specific domains or scenarios. This lightweight approach removes the need for an explicit mediation middleware, and resembles the traditional idea of SOA (where clients and services interact directly) more closely. The practical advantage of client-side mediation is that all context information which may be needed is readily available (e.g., which interface or service the client actually expected, or the format that the client expects the invocation result to be in). Additionally, clients are enabled to construct their individual mediation strategy (by assembling an individual chain of mediators), without relying on any specific support from the service infrastructure.

The contribution of this paper is threefold: firstly, we summarize the general concepts of service mediation; secondly, we present how the existing DAIOS Web service invocation framework [5] has been extended to include a dynamic mediator interface, and thirdly, we explain the implementation of two example mediators that demonstrate the capabilities of this interface. The rest of this paper is structured as follows: Section 2 clarifies the need for invocation-level mediation based on an illustrative example, Section 3 explains the general concepts of mediation, Section 4 details the DAIOS mediation interface and the mediators that we have implemented using this interface, and Section 5 shows the results of a preliminary evaluation. Section 6 elaborates on some related work in the field. Section 7 finally summarizes the paper, and provides an outlook on future work.

2 Motivating Example

To illustrate the need for runtime mediation we present a simple motivating example. Consider the problem of building a composite service for *cell phone number portability*. Number porting is a service pushed by the European Union that allows clients to take their mobile telephone number with them if they change their cell phone operator (CPO). The number porting related business process of a CPO may look roughly as sketched in Figure 1 (simplified for clarity).

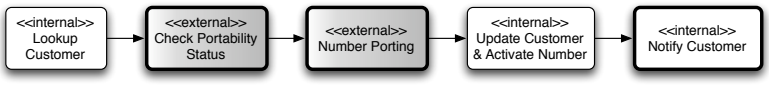


Fig. 1: Number Porting Process

The process starts by looking up the customer using the CPO-internal `Lookup Customer` service. After finding the customer, the process has to send a message to the customer's former CPO to check the portability status. If, for some reason, the porting is not possible, the process is terminated and rescheduled to be executed at a later point (not shown in Figure 1 for brevity). After the portability

check, a request is sent to the old CPO to release the number and transfer it. Afterwards, the account of the customer is updated. Finally, the customer is notified (via SMS, Mail, etc. . .) that the porting is finished.

In this process, only the activities `Lookup Customer` and `Update Customer & Activate Number` are provided by internal services, which can be assumed to have stable and relatively fixed interfaces. The activities `Check Portability Status` and `Number Porting` have to be carried out by external services provided by the CPO that the number has to be ported from. Lastly, `Notify Customer` is an internal activity, which may be provided by a variety of services made available by different internal departments (e.g., by SMS, e-mail, or mail services). This scenario illustrates how essential dynamic adaption is: in some cases the services to invoke differ between instances of the same business process; in other cases the ability to dynamically exchange service providers simply adds value to the process by increasing overall flexibility. Of course it would be possible in this scenario to use e.g., WS-BPEL `Switch` and `Assign` statements to select the appropriate partner service and explicitly reformat the invocation input and output data according to the respective target service, but this approach unnecessarily complicates the business process by shifting what is essentially an implementation issue (selecting the right service provider in a given process instance) to the business process. Additionally, this approach would only scale to a small and well-known number of alternate service providers – if the number of alternatives is very large, or if the alternatives change frequently this workaround quickly becomes unfeasible. Even worse, if the service to invoke has to be looked up dynamically in a service registry this transformation approach fails at any rate. However, note that the actual process of selecting a target service from a set of possible choices is outside the scope of this paper. We tackle this problem within our SOA infrastructure VRESCo [6, 7].

3 Interface-Level Invocation Mediation

Generally, mediation can happen on two different levels: invocation-level mediation defines the mapping of messages (single invocations) between services, while protocol-level mediation considers resolving incompatibilities in the business protocol (invocation ordering) of services. Similar distinctions have previously been identified by different researchers (among others [3, 4, 8, 9]). Per definition, protocol-level mediation is only important for stateful services, since stateless services do not rely on a specific ordering of invocations. Given that SOA traditionally focuses on stateless services we do not cover protocol-level mediation in this paper. However, others have already provided some interesting work in this area (e.g., [10, 11]).

Before going on to explain our mediation architecture, we need to define a number of general concepts that we are going to use in the forthcoming sections. Where applicable, we will use well-known terms (e.g., from the Semantic Web Services (SWS) community [12]) instead of inventing new ones. First of all, we have to distinguish between two different *formats*, *high-level (domain) concepts*

and *proprietary (low-level) formats*. High-level concepts represent things and ideas that exist in the real world, i.e., which are independent from a concrete service or implementation. High-level concepts may (but do not necessarily need to) be concepts in a Semantic Web ontology [13, 14]. Domain concepts are what domain experts talk about. Proprietary formats, on the other hand, are concrete implementations of high-level concepts. They are optimized towards concrete implementation goals, and are specific to single services. In general, proprietary formats motivate mediation – in the end, invocation-level mediation is the process of mediating between different low-level formats that implement the same domain concepts. Mediation between services is only reasonable, if the services implement the same concepts, even though they are probably using different low-level formats to represent them. The general operation of invocation-level mediation is the *transformation* of one format into another. We can distinguish three different types of transformation: (1) transforming high-level concepts into a low-level format is called *lowering* [15]; (2) the inverse operation, transforming proprietary formats into domain concepts is called *lifting*; and (3) we refer to the direct transformation of one proprietary format into another as *conversion*. These general concepts and their relationships are summarized in Figure 2.

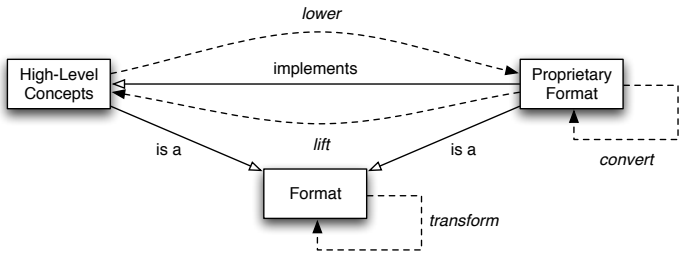


Fig. 2: General Mediation Concepts

We can distinguish three different scenarios for invocation-level mediation (Figure 3). In Scenario (a), a client expects a concrete service interface, but is actually invoking a different one. The invocation is mediated by converting the low-level format provided by the client directly to the format expected by the actual target service. Scenario (b) is similar, but in this scenario mediation is a two-step procedure. Firstly, the client invocation is lifted to domain concepts. Afterwards, this general representation is lowered to the proprietary format expected by the actual target service. The response is processed analogously. Finally, in Scenario (c) the client does not provide the service input in a proprietary format, but already in the conceptual high-level representation. Obviously, this scenario is a special case of Scenario (b) – in this case the processing is simpler since no lifting of the input and no lowering of the response is necessary.

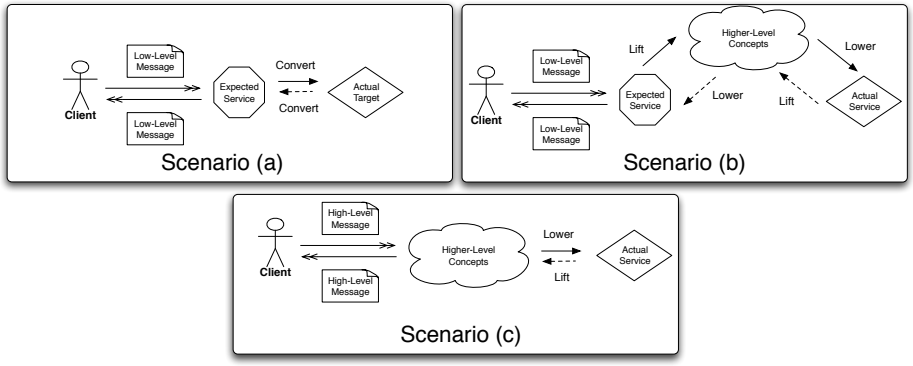


Fig. 3: Mediation Scenarios

These three scenarios are similar from an implementation point of view, but conceptually different. The first two scenarios are typical for legacy clients, or clients that invoke a specific well-known service instance "most of the time", but still need to invoke other services with different contracts from time to time. Speaking in terms of the example from Section 2, one can image that a client for the activity `Notify Customer` was implemented targeting a short message service (since this is the usual way of notifying customers), but still needs to use an e-mail service from time to time. The third scenario is characteristic for clients that have already been built with dynamic binding and runtime service selection in mind. In our example we can assume that clients for the activities `Check Portability Status` and `Number Porting` are implemented in such a way, since there is no "default" service that has to be used more often than others – in these cases, the service to use is entirely dependent on the concrete process instance. In the first scenario, no explicit high-level conceptual representation has to be available. This eases the general mediation model, but scales only to a very small number of possible service alternatives, while the Scenarios (b) and (c) are also applicable to a higher number of alternatives.

4 Mediation Adapters

In this section we will detail how client-side mediation has been implemented within the Daios [5] project. The general idea of Daios is to decouple clients from the services they are invoking by abstracting from service implementation issues such as encoding styles, operations or endpoints. Therefore, clients only need to know the address of the WSDL interface describing the target service and the input message that should be passed to it; all other details of the target service implementation are handled transparently. In Daios, data flowing into and out of services are represented by specific data structures (`DaiosMessage`s). These messages are on a higher level of abstraction than e.g., SOAP messages, and can be represented as an unordered labelled tree structure.

4.1 Daios Invocation Mediation

Even though Daios decouples clients and service providers, the clients still need to know the exact structure of the message that the target service expects. This data coupling is problematic. Services from different providers will usually not rely on the same data model, even if their functionality is equivalent or similar. Therefore, we have extended Daios to include an interface that can be used to hook a *chain of mediators* into the client. The chain of mediators implements a stepwise transformation from the original input (which may be in the proprietary format of a different service, or directly representing high-level concepts) to the proprietary format expected by the target service. Input usually enters the chain encoded as `DaiosMessage`, and the output of the chain is SOAP. Therefore, the mediator chain should at some point contain the "default" mediator, which implements the mapping of the Daios-internal message format to SOAP.

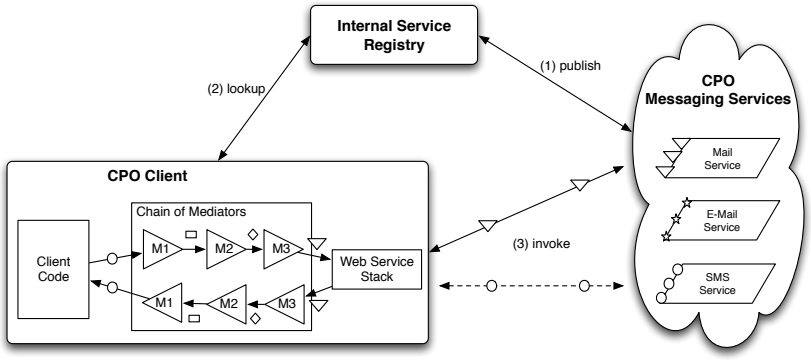


Fig. 4: Client-Side Mediation Architecture

Figure 4 sketches Daios' overall mediation architecture, and how it leverages the standard SOA model of service providers, consumers and registry. In the figure, we exemplify the mediation model based on the activity `Notify Customer` from the process in Figure 1. Additionally, we assume that the client has been developed according to the mediation Scenario (b) from Section 3. (1) A number of different messaging services are published in the service registry. The messaging services all have a similar domain purpose (sending messages to customers), but they are provided by different internal departments of the CPO and are accessible using different interfaces[1]. (2) When the activity `Notify Customer` in the process has to be carried out, the client looks up a messaging service in the registry (according to the preferences of the customer) and constructs a Daios

[1] Note that we do not assume a public service registry. Instead, we work on the assumption of a company-private service registry containing only well-known services, since such registries are more common in today's service-based systems.

frontend to the service (this is a completely automated process that mainly includes parsing the WSDL description of the service and its XML Schema type system, for more details refer to [5]). (3) Finally, the client constructs a message in the proprietary format of one of the possible alternatives (the SMS service in the example) and commences the invocation. The message is now passed through the mediation chain of this client, and will be lifted to a common domain representation using well-defined transformation rules, and again lowered to the format expected by the actual target service. As a last step, the message is serialized to SOAP. This SOAP message is then passed to a Web service stack and sent to the target service. If a return message is received as a response to the invocation, it travels through the mediator chain in the opposite direction, and is passed back to the client in the proprietary format of the SMS service.

In the end, it is the decision of the service client how to construct the mediation chain for every given invocation (i.e., which mediators should be used, and in which order). To do that, the client can choose from a set of general-purpose and domain-specific mediators. Using domain-specific mediators existing domain or service mapping knowledge can be re-used. That decision involves significant knowledge about the services that are likely to be invoked, therefore, a fully automated solution to the mediator selection problem is rather problematic. Mediators may often be able to judge if their application makes sense in a given invocation scenario (e.g., a semantic mediator can judge if semantic annotations are available), but they cannot decide if their application actually resolves all differences in the best way. In our current solution we rely on semi-automated decision making (including humans) in order to construct the mediation chain for any given client. We leave the problem of automatically constructing mediator chains in an optimal way (a problem which bears some resemblance to automated service composition [16]) for our future work.

In the following sections we will detail the implementation of two very different general-purpose mediators, which demonstrate the flexibility of our approach.

4.2 Structural Mediation

One common source for incompatibilities between services is the structure of information. A simple example is sketched in Figure 5, which shows the service interfaces of two different `check_porting_status` services (in DAIOS notion, i.e., as unordered labelled trees). The core information (telephone number, customer identifier, name, location) is contained in both interfaces, but structured differently. Additionally, both interfaces contain a number of fields which are not used in the other message. Consider the case where the user provides input such as the first one in the figure, and the service provides an interface such as the second one. In this case, the incompatibility can be resolved by stepwise transformation of the original user input (i.e., adding new nodes, removing not used nodes, renaming nodes) until the input has the same structure as the service interface. Obviously, information that is not existent in the original message will be left empty in the result message; information that has been present in the original message but not in the service interface is lost. Therefore, the resulting

message is guaranteed to be structurally valid, however, there is no guarantee that the resulting message does not miss mandatory information.

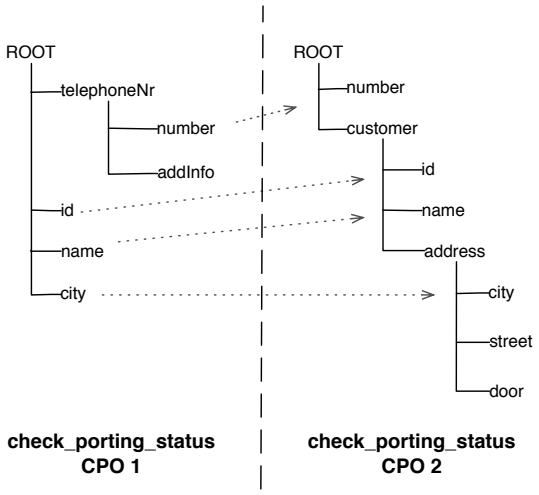


Fig. 5: Structural Interface Mediation

As part of our Daios prototype we have developed a general-purpose mediator that implements such a structural mediation. In terms of the concepts introduced in Section 3 this mediator can implement either Scenario (a) or (c). The mediator is universal, since it does not depend on any additional information besides the target WSDL contract and the client input, and its functionality is well suited to resolve "simple" interface differences (e.g., typical service version changes as introduced in [7]). However, the mediator has a few distinct disadvantages: essentially, the problem of finding the optimal changes to transform a given input tree to a given target format results in computing the tree edit distance (and the respective edit script, i.e., an ordered list of changes that have to be applied), which is known to be NP-hard [17]. The efficiency of the transformation can be improved by implementation-level techniques such as sub-result caching or pruning of identical subtrees. We also cache the edit script associated with every given combination of input and output trees to speed up future similar transformations, however, the transformation effort for unknown invocations still increases exponentially with the amount of change necessary.

In the description above we simplified by assuming that message fields have the same semantics iff they have the same name. In practice, this assumption only holds in rather regulated and controlled environments. In the general case, further data heterogeneity problems arise [18] (such as two fields carrying the same name, but having different semantics). Therefore, it is possible to combine the structural mediator with the semantic mediator (see below) if semantic

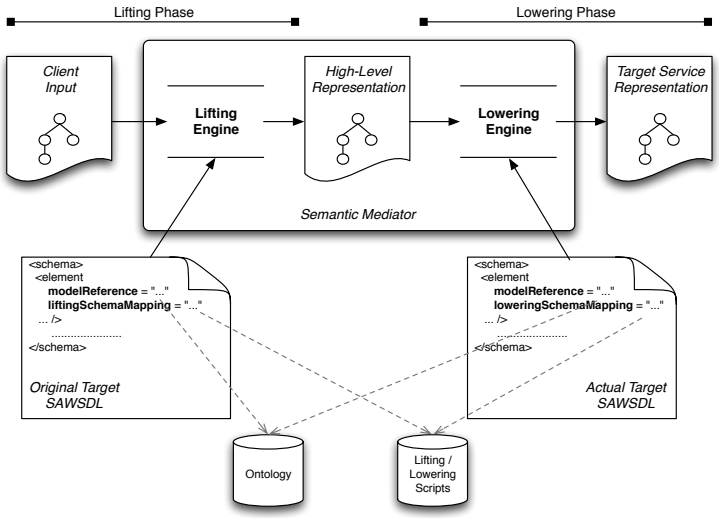


Fig. 6: Semantic Mediation

annotations are available, and use the convention that two message fields have an identical name iff they point to the same ontology concept. Handling these problems without falling back to semantic annotations is part of our future work.

4.3 Semantic Mediation

Currently, most work on Web service incompatibility resolution is carried out within the SWS community. One specifically interesting approach is SAWSDL [19, 20]. SAWSDL provides extensions for WSDL that allows to annotate XML Schema data types, operations and faults with pointers to ontology concepts. Additionally, pointers to scripts that implement lifting and lowering for data types can be added. We have implemented a general-purpose mediator that uses this semantic information in order to implement dynamic mediation between two SAWSDL-annotated Web services. In our implementation, both high-level concepts and proprietary formats are represented using `DaiosMessage`s. Mediation rules are given as transformation scripts.

Figure 6 sketches the general architecture of the Semantic Mediator in a Scenario (b) invocation: the client passes the service input, the SAWSDL description of the original target service and the SAWSDL description of the actual invocation target to the mediator, which retrieves the ontology pointers from the original SAWSDL description, and applies the corresponding lifting scripts; the resulting high-level representation is then transformed to the format expected by the target service by applying the respective lowering scripts (as denoted in the actual target's SAWSDL description). A possible response is processed equivalently (not shown in the figure). Similarly, it is also possible to use the

Semantic Mediator in Scenario (c) situations. In that case, the input provided by the client needs to be annotated with semantic information (i.e., the nodes in the `DaiosMessage` tree need to be annotated with ontology pointers). However, since the input is already provided as a high-level representation, no lifting is necessary and processing starts in the lowering phase (see Figure 6).

In Listing 1, we present an example of a transformation (lowering) script, which leverages the interpreted language Groovy[2]. The script is standard Groovy code, however, we use two special variables (`input` and `output`), which are injected into the interpreter environment (i.e., these variables are already predefined when the execution of the script starts). `Input` refers to the input that is given to the transformation, while `output` represents the transformation result.

```
1  import at.ac.tuwien.infosys.dsg.daiosPlugins.sawsdl.SemanticMessage
2
3  // map telephone number to a complex type in the output
4  ontoUri =
5    "http://infosys.tuwien.ac.at/ontology/nrPorting/data#telephoneNr"
6  newTel = new SemanticMessage()
7  newTel.setString("number", input.getStringByConcept(ontoUri))
8  output.setComplex("telephone_nr",  newTel)
9
10 // map date to a split string in the output
11 ontoUri =
12    "http://infosys.tuwien.ac.at/ontology/nrPorting/data#date"
13 merged = input.getStringByConcept(ontoUri)
14 if(merged != null) {
15
16   year = Integer.parseInt(merged.substring(0,4))
17   month = Integer.parseInt(merged.substring(6,7))
18   day = Integer.parseInt(merged.substring(9))
19   newDate = new SemanticMessage()
20   newDate.setInt("day",day)
21   newDate.setInt("month",month)
22   newDate.setInt("year",year)
23   output.setComplex("porting_date", newDate)
24
25 }
```

Listing 1: Lowering Script Example

In the listing, two values are mapped: on the one hand, on lines 4 to 8, a rather simple mapping of a value (identified by the URI `http://infosys.tuwien.ac.at/ontology/nrPorting/data#telephoneNr`) to a slightly nested data structure (`telephone_nr/number`). On the other hand, on lines 11 to 23, a more complex transformation which splits the value of `http://infosys.tuwien.ac.at/ontology/nrPorting/data#date` into three separate message fields, `porting_date/day`, `porting_date/month` and `porting_date/year`.

Since transformation rules are defined using standard Groovy scripts arbitrary complex transformations may be defined. However, it is not mandatory to use Groovy: since our semantic mediator is based on the the Java 6 scripting

[2] `http://groovy.codehaus.org/`

engine (`javax.script`), transformation scripts can be written in many different interpreted languages (however, we use Groovy by default due to its tight integration with the Java runtime environment). If a different scripting language should be used a new `TransformationFactory` needs to be introduced. The transformation factory implements the loading of the correct scripting engine, and the execution of the script. Listing 2 exemplifies how a new Jython[3] script interpreter can be introduced into the semantic mediation engine.

```
1 TransformationFactory.transformation.put(
2   "application/jython",  // transformations are bound to MIME types
3   JythonTransformer.class  // subclasses TransformationFactory
4 );
```

Listing 2: Integrating New Transformation Engines

The semantic mediator is powerful, however, it depends on the availability of semantically annotated WSDL descriptions, which are not widespread today. Additionally, the script-based approach utilized in our semantic mediator introduces a certain amount of processing overhead (see Section 5). We argue that semantic mediation is best used in cases which demand for extensive semantic transformations, e.g., the integration of existing legacy applications into service-based systems.

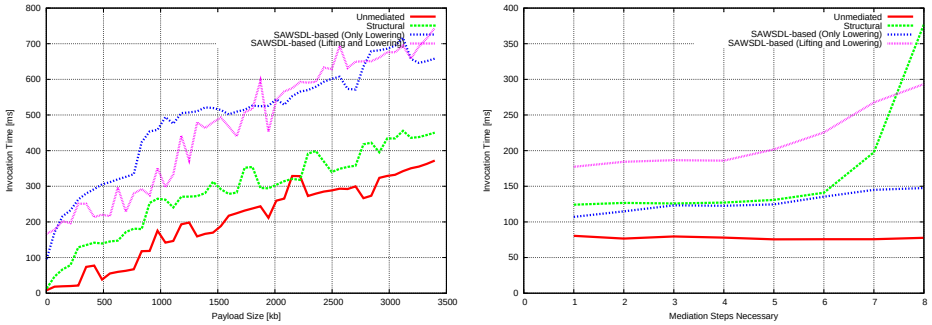
5 Evaluation

We consider runtime performance to be one of the most critical factors of a Web service mediation framework. Therefore, we have carried out a number of performance tests to learn about the processing overhead introduced by the two mediators presented in Section 4. We have compared the performance of unmediated invocations and invocations using the structural and semantic mediator. For the semantic mediator we have evaluated two different scenarios, one resembling Scenario (b) (both lifting and lowering of messages is necessary) and one resembling Scenario (c) (only lowering is necessary). All tests have been carried out on a 2.4 GHz Intel Core 2 Duo machine, with both test clients and services running on the same machine. Every test has been repeated 100 times, and results have been averaged. We summarize the outcomes of this evaluation in Figure 7.

Figure 7a depicts how the invocation time increases with increasing SOAP message payload. Interpreting the figure, we can see that the invocation time increases proportionally with the payload for unmediated invocations (this finding is in line with the results we have already presented in [5]) and all types of mediators. This means that the actual mediation overhead of all mediators is relatively

[3] `http://www.jython.org/Project/`

independent of the payload size. However, we can also see that specifically the semantic mediator introduces a sizable overhead.



(a) (b)

Fig. 7: Runtime Performance Comparison

Obviously, the concrete overhead introduced depends on the amount of mediation necessary. This is depicted in Figure 7b. Here, we plot the invocation time depending on the number of incompatibilities that have to be resolved (e.g., adding one field to an input message). The figure shows that the overhead introduced by the semantic mediator is increasing rather slowly – the bigger part of the overhead is the constant time necessary to load the lifting and lowering scripts, and launch the respective scripting engine. Here, additional tuning might significantly improve the overall performance. The overhead of the structural mediator, on the other hand, varies significantly with the number of incompatibilities: for a small number the mediator introduces practically no overhead, but starting with a certain amount of change the invocation time rises exponentially. This is due to the NP-hardness of the underlying calculation of the tree edit script (see Section 4). In the future, we plan to improve this specific mediator using a heuristic tree edit distance algorithm instead of the currently used deterministic one. However, keep in mind that these results are only valid for the "first" invocation using a given input – because of caching the structural mediator does not inflict a noteworthy delay on any following invocation with the same structure of this client.

6 Related Work

Most related work in the area of resolving interface incompatibilities promotes adapter-based approaches [9, 21]. These adapters are conceptually similar to our mediators, but are more decoupled from the actual clients. We consider these approaches valuable, but think that our approach is more in line with the traditional idea of SOA where clients and providers interact directly. Additionally, our

work allows for simple integration of more complex mediation scenarios, such as mediation based on service semantics (which is hard to accomplish independently from the client). Within the grid computing community, a syntactic mediation approach similar to ours has been proposed [22]. This work uses an ontology-based mediation approach for grid services. Integration of domain-specific mediation knowledge, or structural mediation without semantic information is not covered. To the best of our knowledge no flexible integrated interface mediation framework for general Web services environments like ours has been presented so far.

Other related work has studied mediation on business protocol level. In [3], a number of protocol mismatch patterns are identified, and possible solutions are proposed. In [10], Dumas et al. propose a visual notation and a set of operators that can be used to resolve business protocol incompatibilities. In current industry solutions, service mediation is often handled at ESB level [2]. However, the mediation capabilities of current ESBs such as Apache ServiceMix[4] are limited: from the scenarios presented in Section 3 only Scenario (a) is supported (direct transformation, e.g., applying an XSLT stylesheet to SOAP messages). Service mediation is an often discussed use case for semantic Web services. However, most related work in this community focuses on business protocol incompatibilities. One example is the WSDF framework [23], which uses an RDF model to resolve protocol incompatibilities. Similar research has also been presented in [11]. As part of their work on WSMX, Cimpian et al. have employed the Web Service Modeling Ontology (WSMO) for service mediation [8]. Unlike most other related work, they consider semantic mediation on both interface and process level. These semantic approaches rely on the existence of shared ontologies and explicit semantic information. Even though the same restriction applies for the semantic mediator presented here, our general mediation interface can perfectly be used with other (e.g., domain- or service-specific) mediators when no additional semantic information is available.Other authors have considered mediation on service composition level, e.g., in [24], WS-BPEL processes are adapted by exchanging service bindings at runtime, and compatibility between services is ensured using XSLT-based transformation. Others use annotated WSDL with context information to mediate semantic Web service compositions [25].

7 Conclusion

Currently, dynamic selection of services in SOA-based systems is severely limited by incompatibilities in the interfaces of these services. Enterprise integration solutions such as ESBs or mediation middleware can be used to resolve these problems, but these solutions add additional layers and complexity to the systems built. In this paper we have presented a flexible mediation architecture that enables clients themselves to adapt to varying service interfaces. We have explained the general concepts of interface-level mediation, and how these concepts have been implemented within our existing DAIOS project. The implementation of two

[4] `http://servicemix.apache.org/`

conceptionally different mediators has been used to demonstrate the flexibility of our approach. Additionally, we have discussed the performance overhead introduced by these mediators. Furthermore, we have critically assessed the benefits and limitations of these mediators. Given that there is no single mediator that is able to handle every situation, the incorporation of domain-specific mediators is possible. Additionally, it is possible to combine a number of different approaches in a chain of mediators.

As part of our future work, we plan to extend the work presented in this paper in various directions: firstly, we envision to extend our approach also to mediation on protocol level; secondly, we are aiming at aligning Daios more closely with our SOA runtime environment VRESCo [6, 7], in order to provide an end-to-end SOA environment to clients, and to allow the usage of the VRESCo metadata [26] model for invocation mediation; thirdly, we want to improve the implementation of our general-purpose mediators, e.g., develop a heuristic algorithm to speed up the structural mediator; and lastly, we plan to carry out some work in the area of automated mediator chain construction, i.e., automated determination of a sequence of mediators that is best suited to resolve a given set of incompatibilities.

Acknowledgements

We would like to thank Florian Rosenberg for many helpful discussions and reviews of this paper. The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 215483 (S-Cube).

References

1. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer **11** (2007)
2. Schmidt, M.T., Hutchison, B., Lambros, P., Phippen, R.: The Enterprise Service Bus: Making Service-Oriented Architecture Real. IBM Systems Journal **44** (2005)
3. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M., Toumani, F.: Developing Adapters for Web Services Integration. In: Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE). (2005)
4. Stollberg, M., Cimpian, E., Mocan, A., Fensel, D.: A Semantic Web Mediation Architecture. In: CSWWS. Volume 2 of Semantic Web And Beyond Computing for Human Experience. (2006)
5. Leitner, P., Rosenberg, F., Dustdar, S.: Daios – Efficient Dynamic Web Service Invocation. to appear in IEEE Internet Computing (2009)
6. Michlmayr, A., Rosenberg, F., Platzer, C., Dustdar, S.: Towards Recovering the Broken SOA Triangle – A Software Engineering Perspective. In: Proceedings of the International Workshop on Service Oriented Software Engineering (IW-SOSWE). (2007)
7. Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: End-to-End Versioning Support for Web Services. In: Proceedings of the International Conference on Services Computing (SCC). (2008)

8. Cimpian, E., Mocan, A., Stollberg, M.: Mediation Enabled Semantic Web Services Usage. In: Proceedings of the Asian Semantic Web Conference (ASWC). (2006)
9. Lin, B., Gu, N., Li, Q.: A Requester-Based Mediation Framework for Dynamic Invocation of Web Services. In: Proceedings of the International Conference on Services Computing (SCC). (2006)
10. Dumas, M., Spork, M., Wang, K.: Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation. In: Proceedings of the International Conference Business Process Management (BPM). (2006)
11. Williams, S.K., Battle, S.A., Cuadrado, J.E.: Protocol Mediation for Adaptation in Semantic Web Services. In: Proceedings of the European Semantic Web Conference (ESWC). (2006)
12. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems **16** (2001)
13. Maedche, A., Staab, S.: Ontology Learning for the Semantic Web. IEEE Intelligent Systems **16** (2001) 72–79
14. Pulido, J.R.G., Ruiz, M.A.G., Herrera, R., Cabello, E., Legrand, S., Elliman, D.: Ontology Languages for the Semantic Web: A Never Completely Updated Review. Knowledge-Based Systems **19** (2006) 489–497
15. Kopecky, J., Roman, D., Moran, M., Fensel, D.: Semantic Web Services Grounding. In: Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW '06), Washington, DC, USA, IEEE Computer Society (2006) 127
16. Jinghai Rao and Xiaomeng Su: A Survey of Automated Web Service Composition Methods. In: Proceedings of First International Workshop on Semantic Web Services and Web Process Composition, Springer-Verlag (2004) 43–54
17. Bille, P.: A Survey on Tree Edit Distance and Related Problems. Theoretical Computer Science **337** (2005)
18. Kim, W., Seo, J.: Classifying schematic and data heterogeneity in multidatabase systems. Computer **24** (1991) 12–18
19. World Wide Web Consortium (W3C): Semantic Annotations for WSDL and XML Schema. (2007) `http://www.w3.org/TR/sawsdl/` (Last accessed: April 15, 2008).
20. Kopecky, J., Vitvar, T., Bournez, C., Farrell, J.: SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE Internet Computing **11** (2007) 60–67
21. Cavallaro, L., Di Nitto, E.: An Approach to Adapt Service Requests to Actual Service Interfaces. In: Proceedings of the International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). (2008)
22. Szomszor, M., Payne, T.R., Moreau, L.: Automated Syntactic Medation for Web Service Integration. In: Proceedings of the IEEE International Conference on Web Services. (2006)
23. Eberhart, A.: Ad-hoc Invocation of Semantic Web Services. In: Proceedings of the International Conference on Web Services (ICWS). (2004)
24. Moser, O., Rosenberg, F., Dustdar, S.: Non-Intrusive Monitoring and Service Adaptation for WS-BPEL. In: Proceedings of the 17th International Conference on World Wide Web (WWW). (2008)
25. Mrissa, M., Ghedira, C., Benslimane, D., Maamar, Z., Rosenberg, F., Dustdar, S.: A Context-Based Mediation Approach to Compose Semantic Web Services. ACM Transactions on Internet Technology **8** (2007)
26. Rosenberg, F., Leitner, P., Michlmayr, A., Dustdar, S.: Integrated Metadata Support for Web Service Runtimes. In: Proceedings of the Middleware for Web Services Workshop (MWS'08), co-located with the 12th IEEE International EDOC Conference. (2008)

Efficient QoS-aware Service Composition

Mohammad Alrifai and Thomas Risse

L3S Research Center
Leibniz University of Hannover, Germany
{alrifai|risse}@L3S.de

Abstract. Web service composition requests are usually combined with end-to-end QoS requirements, which are specified in terms of non-functional properties (e.g. response time, throughput and price). The goal of QoS-aware service composition is to find the best combination of services such that their aggregated QoS values meet these end-to-end requirements. Local selection techniques are very efficient but fail short in handling global QoS constraints. Global optimization techniques, on the other hand, can handle global constraints, but their poor performance render them inappropriate for applications with dynamic and real-time requirements. In this paper we address this problem and propose a solution that combines global optimization with local selection techniques for achieving a better performance. The proposed solution consists of two steps: first we use mixed integer linear programming (MILP) to find the optimal decomposition of global QoS constraints into local constraints. Second, we use local search to find the best web services that satisfy these local constraints. Unlike existing MILP-based global planning solutions, the size of the MILP model in our case is much smaller and independent on the number of available services, yields faster computation and more scalability. Preliminary experiments have been conducted to evaluate the performance of the proposed solution.

1 Introduction

Industrial practice witnesses a growing interest in the ad-hoc service composition in the areas of supply chain management, accounting, finances, eScience as well as in multimedia applications. With the growing number of the services available within service infrastructures, the composition problem becomes a decision problem on the selection of component services from a set of alternative services that provide the same functionality but differ in quality parameters. In service oriented environments, where deviations from the QoS estimates occur and decisions upon replacing some services has to be taken at run-time (e.g. in multimedia applications), the efficiency of the applied selection mechanism becomes crucial.

Consider for example the personalized multimedia delivery scenario (from [1]) in Figure 1. A smartphone user requests the latest news from a service provider. Available multimedia content includes a news ticker and topical videos available in MPEG 2 only. The news provider has no adaptation capabilities, so additional services are required to serve the user's request: a transcoding service for the multimedia content to fit the target format, a compression service to adapt the content to the wireless link, a text translation service for the ticker, and also a merging service to integrate the ticker with the video

stream for the limited smartphone display. The user request can be associated with some end-to-end QoS requirements (like bandwidth, latency and price). The service composer has to ensure that the aggregated QoS values of the selected services match the user requirements at the start of the execution as well as during the execution. However, dynamic changes due to changes in the QoS requirements (e.g. the user switched to a network with lower bandwidth) or failure of some services (e.g. some of the selected services become unavailable) can occur at run-time. Therefore, a quick response to adaptation requests is important in such applications. Exponential time complexity for an infrastructure service like the composition service is only acceptable if the number of service candidates and constraints are very limited. Already in larger enterprises and even more in open service infrastructures with a few thousands of services the response time for a service composition request could already be out of the real-time requirements.

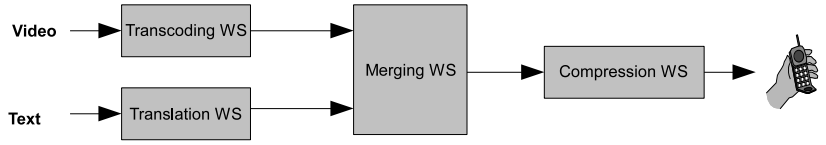


Fig. 1. Composition of Multimedia Web Services

Two general approaches exist for the QoS-aware service selection: local selection and global selection. In local selection, one service is selected from each class independently. Using a given utility function, each service candidate is assigned a utility value and the service with maximum utility value is selected. This approach is very efficient in terms of computation time as the time complexity of the local optimization approach is $O(l)$, where l is the number of service candidates in each class. This is specially useful for distributed environments where central QoS management is not desirable. However, local selection is not suitable for QoS-based service composition, with global end-to-end requirements (like maximum total price), since such global constraints cannot be verified locally.

On the other hand, the global selection [2–5] aims at solving the problem on the composite service level by considering all possible service combinations. The aggregated QoS values of each service combination is computed. This approach seeks the service combination, which maximizes the aggregated utility value, while guaranteeing global constraints. The global selection problem can be modeled as a *Multi-Choice Multidimensional Knapsack problem* (MMKP), which is known to be NP-hard in the strong sense [6]. Therefore it can be expected that an optimal solution may not be found in a reasonable amount of time [7].

Since the business requirements (such as response times, throughput or availability) are only approximate, we argue that finding a reasonable selection of services that covers the requirements "'approximately"' and avoids obvious violations of constraints at acceptable costs is more important than finding "'the optimal"' selection of services with a very high cost. The aim of our study is to find a compromise between optimality

and performance by combining local selection and global optimization to benefit from the advantages of both techniques.

The contribution of this paper can be stated as follows. We divide the global QoS optimization problem into two sub-problems that can be solved separately to improve the efficiency:

- The first sub-problem being the decomposition of global QoS constraints into local constraints, is modeled as a mixed integer linear program [8]. The size of the resulting program is independent on the number of service candidates and hence can be solved more efficiently than existing MILP-based solutions.
- The second sub-problem being the selection of component services is solved by means of local selection. We guide the local selection by means of some global parameters to ensure that the computation of the service utility is not biased by local characteristics.

The rest of the papers is organized as follows. In the next section we review some related work. Section 3 introduces the system model and gives a problem statement. Our approach for efficient QoS computation for web service composition is presented in Section 4. Preliminary empirical results for comparing the performance of our solution with the performance of global planning are presented in Section 5 Finally, Section 6 gives conclusions and an outlook on possible continuations of our work.

2 Related Work

The functional requirements for web service composition can be stated in a workflow language such as Business Process Execution Language (BPEL) [9]. In [10, 11] ontology-based representations for describing QoS properties and requests were proposed to support semantic and dynamic QoS-based discovery of web services. Quality of service management has been widely discussed in the area of middleware systems [12–16]. Most of these works focus on QoS specification and management. Recently, the QoS-based web service selection and composition in service-oriented applications has gained the attention of many researchers [2–5, 17, 18]. In [17] the authors propose an extensible QoS computation model that supports open and fair management of QoS data. The problem of QoS-based composition is not addressed by this work. The work of Zeng at al. [2, 3] focuses on dynamic and quality-driven selection of services. The authors use global planning to find the best service components for the composition. They use (mixed) linear programming techniques [8] to find the optimal selection of component services. Similar to this approach Ardagna et al. [4, 5] extends the linear programming model to include local constraints. Linear programming methods are very effective when the size of the problem is small. However, these methods suffer from poor scalability due to the exponential time complexity of the applied search algorithms [19, 7]. In [18] the authors propose heuristic algorithms that can be used to find a near-to-optimal solution more efficiently than exact solutions. The authors propose two models for the QoS-based service composition problem: 1) a combinatorial model and 2) a graph model. A heuristic algorithm is introduced for each model. The time

complexity of the heuristic algorithm for the combinatorial model (WS_HEU) is polynomial, whereas the complexity of the heuristic algorithm for the graph model (MCSP-K) is exponential. Despite the significant improvement of these algorithms compared to exact solutions, both algorithms do not scale with respect to an increasing number of web services and remain out of the real-time requirements. Any distributed implementation of these algorithms would rise a very high communication cost. The WS_HEU for example, is an improvement of the original heuristic algorithm named M-HEU [20]. The M-HEU algorithm starts with a pre-processing step for finding an initial feasible solution, i.e. a service combination that satisfies all constraints but not necessarily is the best solution. A post-processing step improves the total utility value of the solution with one upgrade followed by one or more downgrades of one of the selected component services. Applying this algorithm in a distributed setting where the QoS data of the different service classes is managed by distributed service brokers would rise very high communication cost among these brokers to find the best composition. In this paper, we propose a heuristic algorithm that solves the composition problem more efficiently and fits well to the distributed environment of web services.

3 System Model and Problem Statement

3.1 Abstract vs. Concrete Composite Services

In our model we assume that we have a universe of web services $\mathbb{S}$ which is defined as a union of *abstract service classes*. Each abstract service class $S_j \in \mathbb{S}$ (e.g. flight booking services) is used to describe a set of functionally-equivalent web services (e.g. Lufthansa and Qantas flight booking web services). In this paper we assume that information about service classes is managed by a set of service brokers as described in [17, 21]. Web services can join and leave service classes at any time by means of a subscription mechanism. We also distinguish between the following two concepts:

– An abstract composite service, which can be defined as an abstract representation of a composition request $CS_{abstract} = \{S_1, \ldots, S_n\}$. $CS_{abstract}$ refers to the required service classes (e.g. flight booking) without referring to any concrete web service (e.g. Qantas flight booking web Service).

– A concrete composite service, which can be defined as an instantiation of an abstract composite service. This can be obtained by bounding each abstract service class in $CS_{abstract}$ to a concrete web service s_j, such that $s_j \in S_j$.

3.2 QoS Vector

In our study we consider quantative non-functional properties of web services, which can be used to describe the quality of a service s. We use the vector $Q_s = \{q_1, q_2, \ldots, q_r\}$ to represent these properties. These can include generic QoS attributes like response time, availability, price, reputation etc, as well as domain-specific QoS attributes like bandwidth, video quality for multimedia web services. The values of these QoS attributes can be either collected from service providers directly (e.g. price), recorded

from previous execution monitoring (e.g. response time) or from user feedbacks (e.g. reputation) [17]. The set of QoS attributes can be divided into two subsets: positive and negative QoS attributes. The values of positive attributes need to be maximized (e.g. throughput and availability), whereas the values of negative attributes need to be minimized (e.g. price and response time). For the sake of simplicity, in this paper we consider only negative attributes (positive attributes can be easily transformed into negative attributes by multiplying their values by -1). We use the function $q_i(s)$ to determine the i-th quality parameter of service s. The QoS information of web services from class S are managed by the responsible service broker of this class.

3.3 QoS Computation of Composite Services

The QoS value of a composite service is decided by the QoS values of its component services as well as the composition model used (e.g. sequential, parallel, conditional and/or loops). In this paper, we focus on the service selection algorithm for QoS-based service composition, and its performance on the sequential composition model. Other models may be reduced or transformed to the sequential model. Techniques for handling multiple execution paths and unfolding loops from [2], can be used for this purpose.

The QoS vector for a composite service CS is defined as $Q_{CS} = \{q'_1(CS), \ldots, q'_r(CS)\}$ where $q'_i(CS)$ represents the estimated QoS values of a composite service CS and can be aggregated from the expected QoS values of its component services. Table 3.3 shows examples of some QoS aggregation functions.

Similar to [2, 17, 5, 18], we assume in our model that QoS aggregation functions either are linear or can be linearized to be represented by the summation relation. For example, QoS attributes that are typically aggregated as a product (e.g. availability) are transformed into a summation relation by applying a logarithm operation.

We extend our model to support the following aggregation function:

$$q'_k(CS) = \sum_{j=1}^{n} q_k(s_j) \tag{1}$$

Table 1. Examples of QoS aggregation functions for composite services

QoS Attribute	Aggregation Function
Response Time	$q'_{res}(CS) = \sum_{j=1}^{n} q_{res}(s_j)$
Price	$q'_{price}(CS) = \sum_{j=1}^{n} q_{price}(s_j)$
Availability	$q'_{av}(CS) = \prod_{j=1}^{n} q_{av}(s_j)$

3.4 Utility Function

In order to evaluate the multi-dimensional quality of a given web service composition a utility function is used. In this paper we use a Multiple Attribute Decision Making approach for the utility function: i.e. the *Simple Additive Weighting (SAW)* technique [22]. The utility computation involves scaling the values of QoS attributes to allow a uniform measurement of the multi-dimensional service qualities independent of their units and ranges. The scaling process is then followed by a weighting process for representing

user priorities and preferences. In the scaling process each QoS attribute value is transformed into a value between 0 and 1, by comparing it with the minimum and maximum possible aggregated value. These values can be easily estimated by aggregating the local minimum (or maximum) possible value of each service class in CS. For example, the maximum execution price of any concrete composite service can be computed by summing up the execution price of the most expensive service in each service class. Formally, we compute the minimum and maximum aggregated value of the k-th QoS attribute as follows:

$$Qmin'(k) = \sum_{j=1}^{n} Qmin(j,k) \text{ and } Qmax'(k) = \sum_{j=1}^{n} Qmax(j,k) \quad (2)$$

where $Qmin(j,k) = \min_{\forall s_{ji} \in S_j} q_k(s_{ji})$ is the minimum value (e.g. minimum price) and $Qmax = \max_{\forall s_{ji} \in S_j} q_k(s_{ji})$ is the maximum value (e.g. maximum price) that can be expected for service class S_j according to the available information about service candidates of this class.

Now the overall utility of a composite service is computed as

$$U'(CS) = \sum_{k=1}^{r} \frac{Qmax'(k) - q'_k(CS)}{Qmax'(k) - Qmin'(k)} \cdot w_k \quad (3)$$

with $w_k \in \mathbb{R}_0^+$ and $\sum_{k=1}^{r} w_k = 1$ being the weight of q'_k to represent user's priorities.

The utility function $U'(CS)$ is used to evaluate a given set of alternative service compositions. However, finding the best composition requires enumerating all possible combinations of service candidates. For a composition request with n service classes and l service candidate per class, there are l^n possible combinations to be examined. Performing exhaustive search can be very expensive in terms of computation time and, therefore, inappropriate for applications with many services and dynamic needs.

3.5 Problem Statement

The problem of finding the best service composition without enumerating all possible combinations is considered as an optimization problem, in which the overall utility value has to be maximized while satisfying all global constraints. Formally, the optimization problem we are addressing can be stated as follows:

For a given abstract composite service $CS_{abstract} = \{S_1, \ldots, S_n\}$ with a set of m global QoS constraints $C' = \{c'_1, \ldots, c'_m\}$, find an implementation $CS = \{s_{1b}, \ldots, s_{nb}\}$ by binding each S_j to a concrete service $s_{jb} \in S_j$ such that:

1. The aggregated QoS values satisfy: $q'_k(CS) \leq c'_k, \forall c'_k \in C'$, and
2. The overall utility $U'(CS)$ is maximized

4 Efficient QoS-aware Service Composition

The use of mixed integer linear programming [8] to solve the QoS-aware service composition problem has been recently proposed by several researchers [2–5]. The decision

(binary) variables in the model represent the service candidates. A service candidate s_{ij} is selected in the optimal composition if its corresponding variable x_{ij} is set to 1 in the solution of the program, and discarded otherwise. The program is formulated as follows (by re-writing (3) to include the decision variables):

maximize the overall utility value given by:

$$\sum_{k=1}^{r} \frac{Qmax'(k) - \sum_{i=1}^{n} \sum_{j=1}^{l} q_k(s_{ji}) * x_{ji}}{Qmax'(k) - Qmin'(k)} \cdot w_k$$

subject to the following global QoS constraints:

$$\sum_{i=1}^{n} \sum_{j=1}^{l} q_k(s_{ji}) * x_{ji} \leq c'_k, 1 \leq k \leq m$$

while satisfying the following allocation constraints on the decision variables:

$$\sum_{j=1}^{l} x_{ji} = 1, 1 \leq i \leq n$$

Because the number of variables in this model depends on the number of service candidates (number of variables = $n * l$), this MILP model may not be solved satisfactorily, except for small instances. To cope with this limitation, we divide the QoS-aware service composition problem into two sub-problems that can be solved more efficiently in two subsequent phases. In the first phase, we use mixed integer linear programming to find the best decomposition of global QoS constraints into local constraints on the component services. The size of the MILP model of this phase is much smaller than the size of the MILP model in [2–5] and can be, therefore, solved much faster. In the second phase, we use local search to find the best component services that satisfy the local constraints from the first phase. The two phases of our approach are described in the next subsections in more details.

4.1 Decomposition of Global QoS Constraints

To ensure that the results of local search comply with the global QoS constraints, we need to set up some local constraints on the QoS values of the individual services. For example, in a composition problem with n service classes, a global constraint such as: total response time ≤ 600 msec, i.e. $\sum_{i=1}^{n} q_{res}(s_i) \leq 600$, needs to be translated into n local constraints in the form of:

$$q_{res}(s_i) \leq R_i, 1 \leq i \leq n \qquad \text{, where} \quad \sum_{i=1}^{n} R_i = 600$$

A naive solution to this problem would be to divide the global constraint into n equal local constraints: $R_i \leq 600/n, 1 \leq i \leq n$. However, as different service classes can have different response times, a more sophisticated decomposition algorithm is required. The

decomposition algorithm need to ensure that the local constraints are not more restrictive than needed in order to avoid discarding any service candidates that might be part of a feasible solution. To solve this problem, we divide the quality range of each QoS attribute into a set of discrete quality values, which we call "*quality levels*". We then use mixed integer linear programming to find the best combination of these quality levels for using them as local constraints.

Quality Levels: In this paper, we use a simple method for constructing the quality levels. We divide the value ranges of each QoS attribute q_k of service class S_i into d levels: $q_{ik}^1, ..., q_{ik}^d$ as follows:

$$q_{ik}^z = \begin{cases} Qmin(i,k) & \text{if } z = 1 \\ q_{ik}^{z-1} + \frac{Qmax(i,k) - Qmin(i,k)}{d} & \text{if } 1 < z < d \\ Qmax(i,k) & \text{if } z = d \end{cases} \quad (4)$$

We then assign each quality level q_{ik}^z a value between 0 and 1, which indicates the probability p_{ik}^z that using this quality level as a local constraint would lead to finding a solution. The probability p_{ik}^z for the zth level of q_k at S_i is computed as follows:

$$p_{ik}^z = h/l \quad (5)$$

where h is the number of service candidates satisfying q_{ik}^z and l is the total number of service candidates at S_i.

MILP Formulation: The goal of our MILP model is to find the best decomposition of QoS constraints into local constraints. Therefore, we use a binary decision variable x_{ik}^z for each local quality level q_{ik}^z such that $x_{ik}^z = 1$ if q_{ik}^z is selected as a local constraint for the QoS attribute q_k at the service class S_i, and $x_{ik}^z = 0$ otherwise. Note that the total number of variables in the model equals to $n * m * d$, i.e. independent on the number of service candidates. By ensuring that the number of quality levels d is small enough such that $m * d \leq l$ we can ensure that the size of our MILP model is smaller than the size of the model used in [2–5].

To ensure that only one quality level is selected from the set of d levels of the QoS attribute q_k at the service class S_i, we add the following set of constraints to the model:

$$\sum_{z=1}^{d} x_{ik}^z = 1, \forall i, \forall k, 1 \leq i \leq n, 1 \leq k \leq m$$

The selection of the local constraints must ensure that global constraints are still satisfied (i.e. the first requirement in 3.5). Therefore, we add the following set of constraints to the model:

$$\sum_{i=1}^{n} \sum_{z=1}^{d} q_{ik}^z * x_{ik}^z \leq c'_k, \forall k, 1 \leq k \leq m$$

The objective function of our MILP model is to maximize the probability that the selected local constraints will lead to finding a feasible composition. Therefore, using (5) the objective function can be expressed as follows:

$$\text{maximize} \quad \bigcap_{i=1}^{n}\bigcap_{k=1}^{m} p_{ik}^{z} \quad \Rightarrow \quad \text{maximize} \quad \prod_{i=1}^{n}\prod_{k=1}^{m} p_{ik}^{z} \quad , 1 \leq z \leq d \qquad (6)$$

Using the logarithmic function to linearize (6) in order to be able to use it in the MILP model, we express the objective functions as follows:

$$\text{maximize} \quad \sum_{i=1}^{n}\sum_{k=1}^{m}\sum_{z=1}^{d} ln(p_{ik}^{z}) * x_{ik}^{z} \qquad (7)$$

By solving this model using MILP solver methods, we get a set of local quality levels that we use in the second phase for guiding local selection.

4.2 Local Selection

The local quality levels, which we obtain from the first phase, are used in the second phase as upper bounds for the QoS values of component services. We filter web services that violate these upper bounds and create a list of qualified services for each service class. The next step in this phase involves sorting the qualified web services by their utility values. We derive a local utility function $U'_{local}(s)$ from $U'(CS)$ that can be applied on the service component's level.

By applying (1) and (2) we get:

$$\begin{aligned} U'(CS) &= \sum_{i=1}^{r} \frac{\sum_{\forall s \in CS} Qmax(class(s), i) - \sum_{\forall s \in CS} q_i(s)}{Qmax'(i) - Qmin'(i)} \cdot w_i \\ &= \sum_{\forall s \in CS} \underbrace{(\sum_{i=1}^{r} \frac{Qmax(class(s), i) - q_i(s)}{Qmax'(i) - Qmin'(i)} \cdot w_i)}_{U'_{local}(s)} = \sum_{\forall s \in CS} U'_{local}(s) \end{aligned} \qquad (8)$$

The utility function $U'_{local}(s)$ can be computed for each service class S_j independently, provided that the global parameters $Qmin'$ and $Qmax'$ are specified. These parameters can be easily computed by beforehand by aggregating the local maximum and minimum values of each service class. Thus, by selecting the service candidate with the maximum U'_{local} value from each class, we can ensure that $U'(CS)$ is maximized (i.e. satisfying the second requirement in 3.5).

5 Experimental Evaluation

In this section we describe preliminary results of experimental evaluation of our proposed solution. We conducted our experiments on a HP ProLiant DL380 G3 machine

with 2 Intel Xeon 2.80GHz processors and 6 GB RAM. The machine is running under Linux (CentOS release 5) and Java 1.6.

In our evaluation we compare the performance of our solution with the performance of the MILP-based "pure" global planning solution [3, 5]. For solving the MILP problems in both approaches we use the open source (Mixed Integer Linear Programming) LpSolve system *lpsolve* version 5.5 [23].

For testing purposes we created 20 service classes with 100 service candidates for each class. The QoS dataset was randomly created by assigning each web service candidate 5 arbitrary values for 5 QoS attributes ranging between 0 and 100. All random values are normally distributed. We created several instances of the QoS composition problem by varying the number of service candidates per class l. The number of global constraints m in all problem instances was fixed to 5 constraints. In the current implementation of our approach we used the simple method for constructing quality levels as described earlier in 4.1. Using this method we divide each quality dimension into 5 quality levels.

Figure 2 shows a comparison of the performance between the "pure" global planning approach (labeled with "Global") and our approach (labeled with "Hybrid") that combines global optimization and local search. We measure the time required by each approach to find the best combination of services (i.e. with maximum utility value) that satisfies all the given global QoS constraints. The measured time does not include any data loading or initialization time as these times are independent of the applied approach. In our evaluation we consider only the pure computational time from the start of until the end of the search process. The results shown are averaged over a total of 100 different runs of the same experiment with the same parameters. In the graph shown on

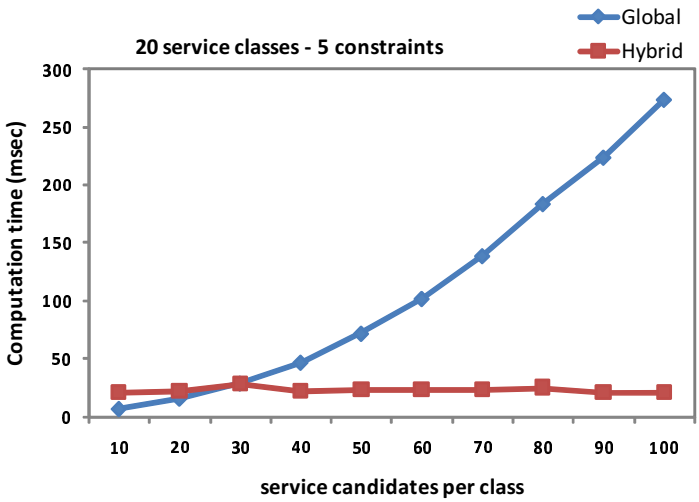


Fig. 2. Computational time with respect to the problem size

Figure 2 we measure the required computation time by each approach with respect to an increasing number of service candidates. The number of service classes n in this experiment is fixed to 20, while the number of service candidates l is varying between 10 and 100 service candidates per class.On the left-side graph we notice that for small-sized

problem instances (with $l \leq 25$) the global planning approach performs better than our solution. This is an expected behavior as we already discussed in Section 4.1. With the number of service candidates $l \leq m * d$ our solution does not gain any improvement in the performance as the size of the used MILP model remains greater than or equal to the original MILP model used by global planning approach. However, with an increasing number of candidates, the computation time of global planning increases dramatically, while the performance of our solution remains unaffected by the number of candidates, which makes our solution more scalable.

6 Conclusion and Current Work

This paper describes an efficient method for the QoS-based service composition. The problem is known to be NP-hard. We combine global optimization with local selection methods to benefit from the advantaged of both worlds. Our proposed method allows to dramatically reduce the (worst case) efforts compared to existing solutions. Preliminary imperial results show a very promising improvement in terms of computational time. This is specially useful for applications with dynamic changes and real-time requirements. Currently we are working on more extensive experiments with larger composition problems. In the work presented in this paper, we use a very simple method for constructing QoS quality levels. We are currently studying the impact of the applied method as well as the number of quality levels used on the performance and quality of the obtained results, especially when different datasets are used.

References

1. Wagner, M., Kellerer, W.: Web services selection for distributed composition of multimedia content. In: MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia, New York, NY, USA, ACM (2004) 104–107
2. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: WWW. (2003) 411–421
3. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Trans. Software Eng **30**(5) (2004) 311–327
4. Ardagna, D., Pernici, B.: Global and local qos constraints guarantee in web service selection. icws **0** (2005) 805–806
5. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Trans. Software Eng. **33**(6) (2007) 369–384
6. Pisinger, D.: Algorithms for Knapsack Problems. PhD thesis, University of Copenhagen, Dept. of Computer Science (1995)
7. Parra-Hernandez, R., Dimopoulos, N.J.: A new heuristic for solving the multichoice multidimensional knapsack problem. IEEE Transactions on Systems, Man, and Cybernetics, Part A **35**(5) (2005) 708–717
8. Nemhauser, G.L., Wolsey, L.A.: Integer and combinatorial optimization. Wiley-Interscience, New York, NY, USA (1988)
9. OASIS: Web services business process execution language (April 2007) `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`.
10. Zhou, C., Chia, L.T., Lee, B.S.: Daml-qos ontology for web services. icws **0** (2004) 472

11. Bilgin, A.S., Singh, M.P.: A daml-based repository for qos-aware semantic web service selection. icws **0** (2004) 368
12. Aurrecoechea, C., Campbell, A.T., Hauw, L.: A survey of qos architectures. Multimedia Systems **6**(3) (1998) 138–151
13. Gillmann, M., Weikum, G., Wonner, W.: Workflow management with service quality guarantees. In: SIGMOD Conference. (2002) 228–239
14. Cui, Y., Nahrstedt, K.: Supporting qos for ubiquitous multimedia service delivery. In: ACM Multimedia. (2001) 461–462
15. Casati, F., Shan, M.C.: Dynamic and adaptive composition of e-services. Inf. Syst **26**(3) (2001) 143–163
16. Issa, H., Assi, C., Debbabi, M.: Qos-aware middleware for web services composition - a qualitative approach. In Bellavista, P., Chen, C.M., Corradi, A., Daneshmand, M., eds.: Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC 2006), 26-29 June 2006, Cagliari, Sardinia, Italy, "IEEE Computer Society (2006) 359–364
17. Liu, Y., Ngu, A.H.H., Zeng, L.: Qos computation and policing in dynamic web service selection. In: WWW. (2004) 66–73
18. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end qos constraints. TWEB **1**(1) (2007)
19. Maros, I.: Computational Techniques of the Simplex Method. Springer (2003)
20. Khan, S., Li, K.F., Manning, E.G., Akbar, M.M.: Solving the knapsack problem for adaptive multimedia systems. Stud. Inform. Univ. **2**(1) (2002) 157–178
21. Li, F., Yang, F., Shuang, K., Su, S.: Q-peer: A decentralized qos registry architecture for web services. In: ICSOC. (2007) 145–156
22. Yoon, K..P., Hwang, C.L.: Multiple Attribute Decision Making: An Introduction (Quantitative Applications in the Social Sciences. Sage Publications (1995)
23. Michel Berkelaar, Kjell Eikland, P.N.: Open source (mixed-integer) linear programming system. Sourceforge `http://lpsolve.sourceforge.net/`.

A Distributed Service Component Framework for Interoperable and Modular Service Oriented Pervasive Computing Applications

Daniel Pakkala and Juho Perälä

VTT Technical Research Centre of Finland, P.O. Box 1100, FI-90571 Oulu, Finland
E-mail: {daniel.pakkala, juho.perala}@vtt.fi

Abstract. This paper contributes a novel service component framework enabling development of interoperable and modular service oriented applications in gateway based pervasive computing environments. The research problem addressed in the paper is how to enable integration, interoperability and modularity of technologically heterogeneous software services in distributed pervasive computing environments. The research problem is addressed by proposing a service component framework introducing an integration layer for heterogeneous service oriented software technologies. The novelty of the proposed solution is its capability to enable interoperable registration, discovery, binding and interaction of software services implemented in multiple different service oriented software technologies. The service component framework presented in the paper is implemented by a middleware platform, known as the MidGate platform, and is functionally validated via a laboratory prototype system implementation overviewed in the paper.

1. Introduction

The ongoing adoption of service oriented computing paradigm, throughout the field of computing, is radically changing the way information, communication and automation systems are being built. In enterprise and Internet computing domains the Web Services [3] technology is becoming a standard applied for developing systems and software [5]. However, in pervasive computing environments, such as digital home and mobile computing environments, the adoption of service oriented computing paradigm has emerged in recent years and there is no all upon agreed standard technology for building service oriented systems. Rather in these environments multiple competing service oriented system and software technologies are emerging due to high resource consumption and inflexibility of the Web Services technology [3] in highly heterogeneous, resource limited and dynamic pervasive computing environments.

Service oriented system and software technologies such as UPnP [15], DPWS [2] and other optimized Web Services specification family based technologies have emerged in the pervasive computing domain. This development poses a challenge for system and software developers of pervasive computing environments forcing developers to choose one of the emerging technologies without interoperability with the rest of

them. This development is also leading towards further fragmentation of software and hardware products in pervasive computing environments from the interoperability viewpoint and restrains service provisioning into pervasive computing environments; There are no development and deployment middleware available that would be able to support coexistence of multiple service oriented system and software technologies in distributed pervasive computing environments.

A common characteristic of the most of the existing service oriented system and service technologies is absence of component models making it challenging to develop modular software applications and services based on these technologies. However, some work has been done on the area of enhancing modularity of service oriented software systems; Technologies such as OSGi [7], [8], iPOJO [6], SCA [1] include a service-component model representing the convergence of service and component oriented software development, which has great potential of producing highly modular and interoperable software products. However, these technologies do not address the modularity and interoperability challenge in distributed and dynamic pervasive computing environment, where runtime adaptation and reconfiguration by migration of service components is required for adaptive workload allocation and context-aware behavior of the system.

In this paper a distributed service component framework, referred to as MidGate Framework is presented. The MidGate Framework is the core part of the MidGate middleware service architecture and platform [10], [11], which are targeted for facilitating application development in distributed and dynamic pervasive computing environments. The component model of the MidGate middleware platform has previously been published with a focus in behavioral adaptation of middleware applications and services [13]. This paper complements the previously published work by presenting the MidGate service component framework enabling development of interoperable and modular software services and applications in gateway based heterogeneous and distributed computing environments.

The structure of the paper is as follows. Section 2 provides an overview of the MidGate Framework and related system architecture. Section 3 presents the MidGate Service Component structure and lifecycle actions within the MidGate Framework. The framework architecture is presented in section 4, followed by section 5 presenting the functional validation of the service component framework via a laboratory prototype system. Finally, conclusions are presented closing the paper.

2. Framework and System Architecture

In pervasive computing environments, such as mobile computing and digital home environments, multiple competing service oriented software technologies are emerging and being developed. This development is leading towards further fragmentation of software and hardware products and requires new software development and deployment environments that are able to ensure both interoperability and modularity of service oriented applications. The MidGate platform [10], [11] and especially it's service component framework presented in this paper, have potential for fulfilling these requirements with a gateway based middleware platform that is able to support coexis-

tence, integration and interoperability of multiple service oriented system and software technologies.

Figure 1 illustrates an example deployment of the distributed MidGate platform on a home gateway and a mobile terminal and an example composite service implementation (Service Z) where the composite services (UPnP Service X and Web Service Y) are implemented using different service oriented software technologies (e.g. UPnP and Web Services). Further in this paper we use term service ecosystem to refer to individual deployed Service Oriented Architectures (SOA) that have their unique mechanisms and technologies for service registration, discovery, binding and interaction interoperable within the same SOA, but not interoperable across the different service oriented architectures.

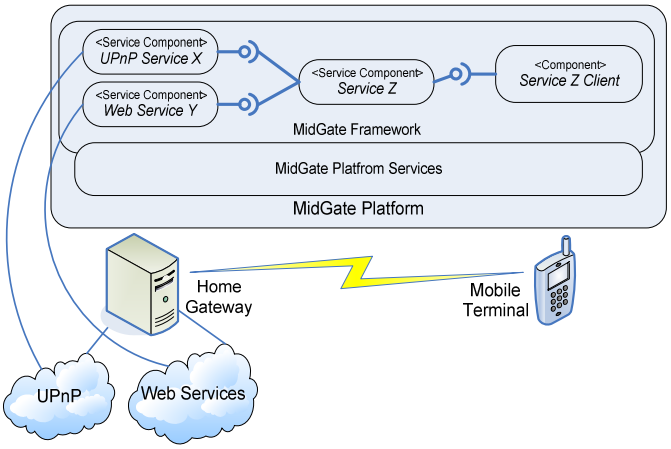


Fig. 1. MidGate Framework coverage.

As illustrated in the figure 1, the MidGate Framework provides an additional service interoperability layer which is able to compose services residing in different service ecosystems, each having their own service oriented architecture and related mechanisms for service registration, discovery, binding and interaction. By providing a complementary distributed service component framework, realizing an additional integrating service oriented architecture, the MidGate platform enables modular integration of multiple different service ecosystems that are not compatible with each other otherwise. The MidGate platform provides two different options to implement the integration with external service ecosystems:

1. **Service proxy** that implements a proxy to a service available in external service ecosystem and registers a corresponding service within the MidGate Framework for access via the service oriented architecture realized by the MidGate platform.
2. **Registry synchronization** where the entries from a service registry (service descriptions and ports) of an external service ecosystem are forwarded and registered automatically towards the MidGate Framework and its internal runtime service registry.

In the service proxy option full interoperability within the MidGate framework, including service registration, discovery, binding and access, is achieved but the ap-

proach is service specific and each external service needs a corresponding proxy component to be implemented at the MidGate platform. Accordingly in highly dynamic and adaptive applications the registry synchronization approach should be favored over the service proxy approach.

As illustrated in figure 1, the MidGate middleware platform, which is exposed to application and service developers via the MidGate Framework, realizes a middleware integration layer able to provide interoperability with services implemented in different service oriented software technologies. Fundamentally, on the implementation level this is enabled by the gateway based deployment and communication approach of the MidGate platform, where the MidGate gateway host that is present in the system should include all the service oriented software stacks utilized in the system. As a single host and non-distributed middleware the OSGi Service Platform is able to provide the same modular integration capability via the device access specification [8] and related base driver scheme. However the MidGate platform and Framework extends this capability into distributed gateway based computing environment instead of a single host. This enables the external service ecosystems to be accessed also from terminals that do not include the software stack of an external service ecosystem. Accordingly for example UPnP technology based services can be accessed from a non UPnP enabled mobile terminals considerably facilitating application development and interoperability for the resource constrained terminals.

Figure 2 presents the MidGate platform's system architecture highlighting the role and capabilities of the MidGate Framework as a service component framework distributed over the host infrastructure of a distributed and gateway based computing environment. In [14] a comprehensive layering of middleware for embedded systems has been presented. In terms of the layering presented in [14] the MidGate platform establishes a middleware platform residing on layers from host infrastructure middleware to domain specific middleware, establishing a middleware platform for development and deployment of interoperable and modular service oriented pervasive computing applications.

As illustrated in figure 2, The MidGate system architecture includes an infrastructural layering of a distributed computing environment. The layers present in the system architecture include *Network*, *Host*, *Middleware* and *Application and Service infrastructure:*

- *Network infrastructure layer* includes all networking technologies that can be seen as enabler for communication between two or more network connected hosts. The layer is further divided to front-end and back-end networks accordingly to the type of host they connect with the gateway host.
- *Host infrastructure layer* includes all hardware, operating system, programming language and virtualization technology that can be seen as enabler for executing software locally on a single host.
- *Middleware infrastructure layer* includes all distribution middleware or middleware service technologies that can be seen as enabler for development of interoperable distributed software in heterogeneous network and host infrastructure environment. The layer is further divided into MidGate middleware services (provided by the platform [10], [11]) and external middleware technologies and services (e.g. Java RMI, Corba, Web Services and UPnP).

- *Application and service infrastructure layer* includes all software or service platform technologies that can be seen as enablers for development and deployment of service oriented computing applications; The exchanged data between software services and applications is structured and typed. The operations for processing data are described and can be registered, discovered and invoked via a shared registry.

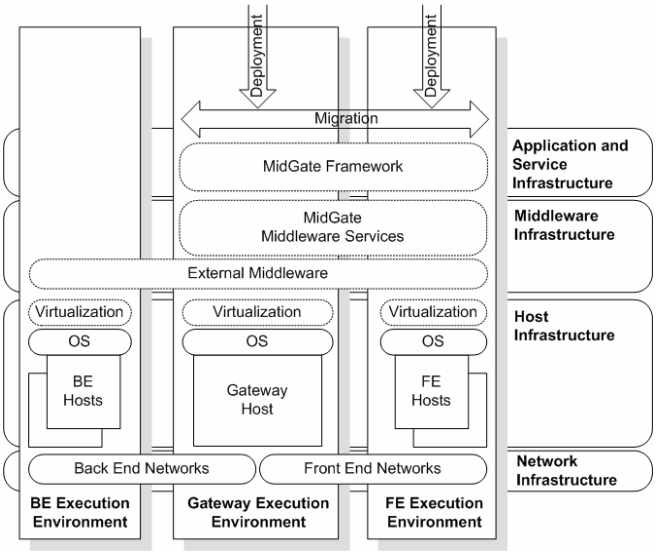


Fig. 2. MidGate system architecture.

As illustrated in figure 2, the MidGate system architecture includes also three different types of execution verticals that represent singular host execution environments in a distributed computing setting: back-end execution environments, gateway execution environment and front-end execution environments. The execution verticals are crosscutting of infrastructure horizontals from the viewpoint of singular host. The concept of execution verticals of the MidGate system architecture is beneficial for the application and service designers by helping to form an overall view of the infrastructural capabilities of a singular execution environment as part of a distributed and heterogeneous system. This enables specification of resources available for deployment units based on the selected execution environment.

As illustrated in Figure 2, the MidGate system model enables deployment of service implementations as service components either at the gateway host (e.g. home gateway) or at the front-end hosts (e.g. mobile terminals). From the viewpoint of software developer the MidGate software APIs, accessed via the MidGate framework at runtime, are identical both at the gateway and front-end hosts. Accordingly the same capabilities and middleware services are provided at both of the aforementioned host types. Further after deployment the MidGate Framework enables runtime migration of service components between the gateway and front-end hosts for components that are independent of any special local host resources. The migration possibility is beneficial e.g. for implementing adaptive workload allocation between the MidGate gateway and front-end hosts (e.g. resource constrained mobile terminals).

As illustrated in Figure 2, the MidGate platform covers only gateway and front-end hosts. Integration to back-end host infrastructure (e.g. to web servers or home automation gateways) is defined to take place via external standard or legacy middleware (e.g. Web Services stack, any Object-oriented middleware or UPnP). Interoperability within the MidGate framework is achieved via the MidGate platforms middleware infrastructure including middleware services known as generic service elements (GSEs) [11]. These include basic middleware services such as messaging, event notification and data management between all MidGate components. More information about MidGate platform and it's middleware infrastructure can be found in [10], [11].

Interoperability with multiple different service oriented software technologies is achieved via the MidGate Framework which in addition to realizing a service component framework for platform's internal middleware services, also provides a capability to register services residing in external service ecosystems. As the capabilities of different front-end hosts may differ in terms of service access technology (e.g. UPnP or Web Services), the MidGate Framework takes this fact into account in the service registration and discovery process.

2.1 Service Registration and Discovery

The MidGate Framework is a service component framework and accordingly manages both the component (e.g. service's software implementation) and the service lifecycle within the platform at runtime. Accordingly, the overall MidGate Framework consists of two sub-frameworks; service framework and component framework both managing the service components from their viewpoint of responsibility. The service registration and discovery process of the MidGate platform is illustrated in Figure 3.

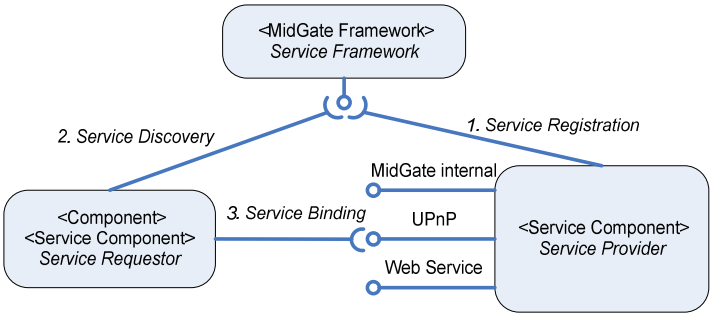


Fig. 3. Service registration and discovery.

As illustrated in Figure 3, the MidGate platform supports two different component types:

- **Service Component** is a software component providing and registering one or more services within the MidGate Framework. This component type interacts with both sub-frameworks of the MidGate Framework; Service Framework and Component Framework.

- **Component** is a software component that is executed on the platform but does not provide or register any services within the MidGate Service Framework. This component type interacts with the component framework and may also utilize the service framework for discovering and utilizing services.

Accordingly, the service implementations on MidGate platform are always of type *Service Component*. The service requestors may be either of type *Component* in case of pure applications and service clients, or of type *Service Component* in case of composite services.

As illustrated in Figure 3, the MidGate Framework supports registration and discovery of service components that have multiple service ports implemented in different service oriented software technologies (e.g. UPnP, Web Services or MidGate platform's internal SOA based on messaging based service interaction [12]). The information of all the available service ports of a service component is included in the service description which is registered to the framework by the service providing component. Accordingly, the MidGate platform enables development and deployment of services having multiple service ports to the same service enabling selection of the most suitable port and corresponding technology for the service requestor. Information about the service interaction capabilities of the host of the service requesting component is included in the discovery process making the framework able to compare the availability of different service ports with the service interaction capabilities of the requestor. As a result only compatible service ports are returned for the service requestor as a result of service discovery process. The guaranteed minimum service access technology throughout the MidGate platform is the platform's internal messaging based service interaction [12]. Additional service ports are mediated by the framework depending on the execution environment capabilities of the service components.

As was illustrated in figure 2, the MidGate Framework is distributed over the gateway and front-end hosts of the system. Accordingly the service registries, encapsulated by the framework, are distributed and synchronized. The main service registry is maintained at the MidGate gateway host and service registrations and discovery requests originating from the front-end hosts are propagated to the main registry at the gateway when connected with the gateway. When a front-end host connects with the gateway host the service registers are automatically synchronized. When the gateway is unreachable by the front-end hosts the local copy of the host's service registry is updated in service registration and searched in service discovery process.

3. Service Component Structure and Lifecycle

As described in section 2.1, the MidGate platform supports two different types of components; *Component* and *Service Component*. The general structure of MidGate component types is illustrated in Figure 4. As illustrated in Figure 4, both MidGate component types consists of three main parts; component implementation, component interfaces and component descriptions. Also both of the component types include application logic, adaptation control, lifecycle control and service control implementations, as well as a component interface and description. However in case of *Service Component* additional parts are included in the structure: service logic implementa-

tion, service interface and service description. Accordingly, the type *Component* is a supertype of *Service Component*. The service and application logic parts of the component implementation represent software implementation of a service and additional application functionality respectively. The adaptation control part of the component implementation is responsible of controlling the components behavior when adaptation requests are given by the MidGate Framework. More information about the adaptive behavior of MidGate components can be found in [13].

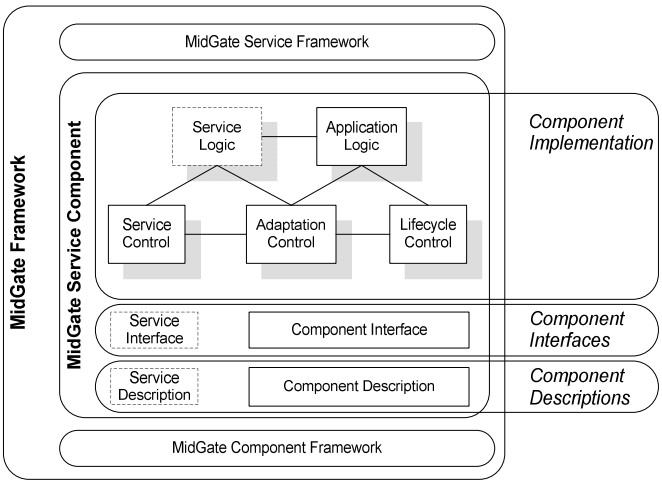


Fig. 4. Service component structure.

The service control part of a service component implementation is responsible of managing the interactions with MidGate Service Framework. This includes management of service description registration, and discovery of possible composite services. The lifecycle control part of a MidGate component is responsible of controlling the behavior of the component as managed by the MidGate Component Framework. Figure 5 illustrates the different lifecycle actions of both MidGate component types and the corresponding component implementation parts involved in a typical service provider – requestor setting.

To illustrate the different lifecycle actions of MidGate components a typical service provider – requestor setting illustrated in Figure 5 is described; A MidGate Service Component (provided service) is deployed into a MidGate Component Framework with a corresponding component description. The component framework instantiates and starts the service component via its component interface and retains the instance as a reference towards a started component for execution control (e.g runtime behavioral adaptation [13]).

Triggered by the start command received from the component framework, the lifecycle control implementation of the *Service Component* makes the necessary control actions to activate the service logic and additional application logic. Once active, the lifecycle control triggers the service control implementation. The service control implementation discovers the composite services needed by the service component via the MidGate Service Framework. Once discovered the service control implementation

binds to composite services and registers the *Service Component's* service description and corresponding service interface to the MidGate Service Framework. Now the service provided by the *Service Component* is available for discovery and consumption via the service framework.

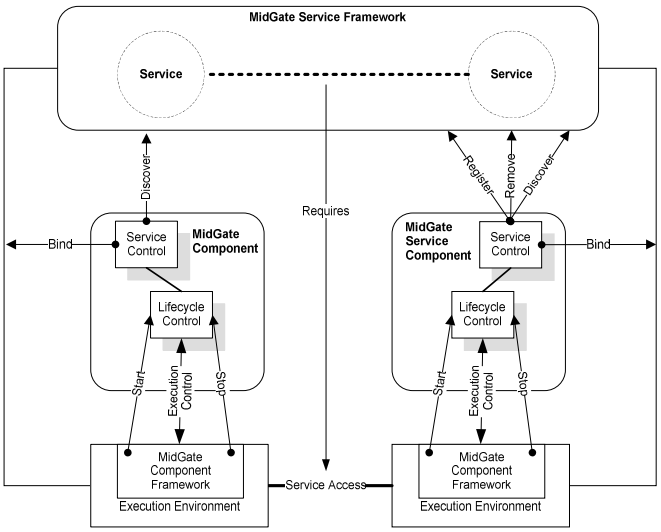


Fig. 5. Service component lifecycle actions.

A MidGate *Component* (service requestor) is deployed into a MidGate Component Framework with a corresponding component description. The MidGate Component Framework instantiates and starts the component via its component interface and retains the instance as a reference towards a started component for execution control. Triggered by the start command, the lifecycle control implementation of the *Component* triggers the service control implementation that discovers and binds the service provided by the *Service Component* via the service framework. After this the lifecycle management makes the necessary control actions to activate the application logic making the actual service request.

The presented setting is only one example and naturally the deployment order of the components does not need to be the one presented as the MidGate Service Framework holds the runtime information of services and their availability. Accordingly a service that is not functional and ready to be used can not be found via the framework.

4. Framework Architecture

The MidGate component architecture has been previously presented in [13]. In this paper we overview the related MidGate Framework architecture. As was already illustrated in Figures 4 and 5, the framework consists of two sub-frameworks; service framework and component framework. The framework architecture has been defined

based on the principle of separation of concerns. In the case of the MidGate Framework the two concerns identified were management of service lifecycle and management of component lifecycle in a distributed computing environment co-operatively. It is important to notice that in terms of functionalities related to service component lifecycle, the MidGate Framework is a runtime framework relevant after component deployment and during their execution. Additional service component repositories can be applied during the development time to achieve higher reusability of the service components developed for the MidGate platform. Figure 6 illustrates the relation between component and service lifecycle within the MidGate Framework.

As illustrated in Figure 6, the service lifecycle is clearly separate, but takes place within the component lifecycle. As the MidGate platform is designed for highly dynamic pervasive computing environments, which require resource-aware behavioral adaptation and runtime reconfiguration of the system, no tight control constraints exist between the component and service lifecycles within the framework. The component lifecycle in the upper part of Figure 6, also presented in detail in [13], consists of 5 different component states and 8 state transfer actions related to management of adaptive component execution within the MidGate platform. As illustrated in figure 6, the service lifecycle within the framework consists of two different states: COMPOSED and ACTIVE. The related state transfer actions are: *compose service, register service* and *remove service*. The *compose service* action includes the discovery and binding of the needed composite services within the framework and all other actions needed in order to activate the service. The *register service* action includes registration of service description and ports of the service component within the framework making the service available for discovery and binding via the framework. The *remove service* action includes removal of the service description and ports from the framework. The mutual ordering of the service states is defined in a way that COMPOSE state must precede the ACTIVE state.

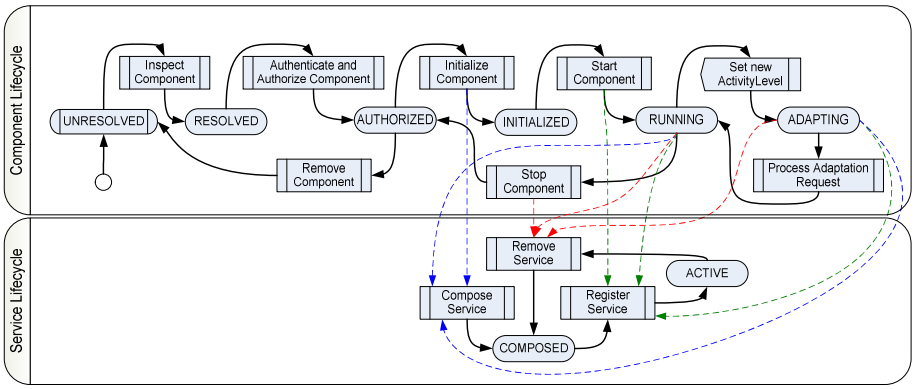


Fig. 6. Service and component lifecycles within the MidGate Framework.

As illustrated in Figure 6, the compose service action can be triggered either in parallel to component initialization or at any time while the component is in active execution in RUNNING or ADAPTING states. Services that are available independently of the runtime context can be composed and registered in parallel to component

initialization and startup. The services which are dependent on runtime context (e.g. dynamically appearing composite services) can be composed and registered any time while the component is in RUNNING state. The services which are reflected by behavioral adaptation of the component (e.g. adaptation to dynamically changing computing environment resources) can be removed and registered in order to manage temporary unavailability of the service while processing the adaptation request received via the MidGate Component Framework. If the component adaptation is not behavioral, but functional or structural adaptation to dynamically changing context information, then also new services may be composed and registered as well as the existing services removed.

As a summary there are no tight control constraints between the component and service lifecycles and accordingly the MidGate Framework architecture is based on separating these two different concerns into two different sub-frameworks of the MidGate Framework: Component Framework and Service Framework. Figure 7 presents and overview of the MidGate Framework architecture including the two sub-frameworks.

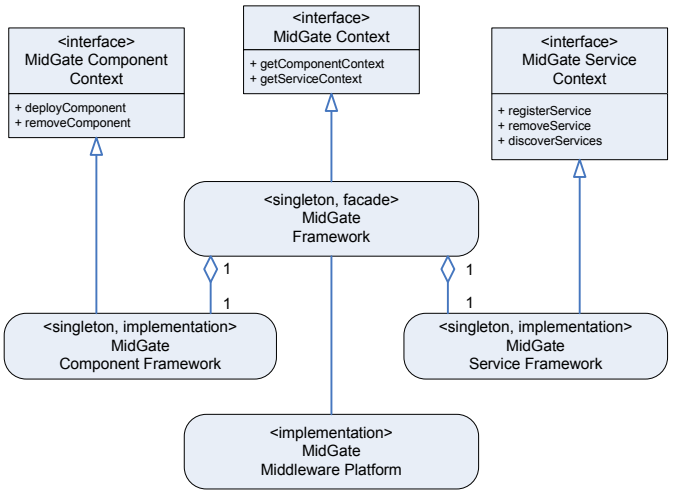


Fig. 7. MidGate Framework architecture.

As illustrated in Figure 7, the MidGate Framework applies the singleton and facade patterns in combination to implement a two dimensional framework that can be accessed via one access point. In terms of interfaces the term context is used to reflect the runtime view of the MidGate Framework:

- **MidGate context**, a single shared runtime object reference that can be used for accessing all the services within the MidGate platform.
- **Service context**, a single shared runtime object reference that can be used for managing the services within the MidGate Service Framework (excluding platform services that are controlled by the platform implementation).
- **Component context**, a single shared runtime object reference that can be used for managing the components within the MidGate component framework.

When implemented the MidGate framework outlook on each host remains the same providing similar capabilities at each deployed host. However, due to the scale down approach of the MidGate architecture [11], enabling deployment to resource constrained hosts, the implementations of the middleware platform and included services may be different depending on available target host resources. Regardless of differences in platform implementations the framework provides an interoperable service integration layer throughout the system as was illustrated in Figure 2.

5. Prototype

A laboratory prototype system was developed for functional validation of MidGate Framework's capability to integrate services from external service ecosystems into a service composition. The application scenario used in the prototype validation was an intelligent alarm system service implemented on the MidGate platform. The alarm system service was controllable by user via a mobile terminal and when activated automatically notified the user about possible burglary at home. The alarm system service was based on three other services: a security camera, motion detection and lighting control services, each residing in a different service ecosystem.

The logic of the alarm system service in short was following; If motion detection event is received from the motion detection service, the lighting control service is invoked to set home lighting on. After this security camera service is invoked to start capture of video and still images from the home. Having the results from the composite services, the alarm system service informs user application with alarm notification including still images and video from security camera service. The laboratory prototype system and its components are illustrated in figure 8.

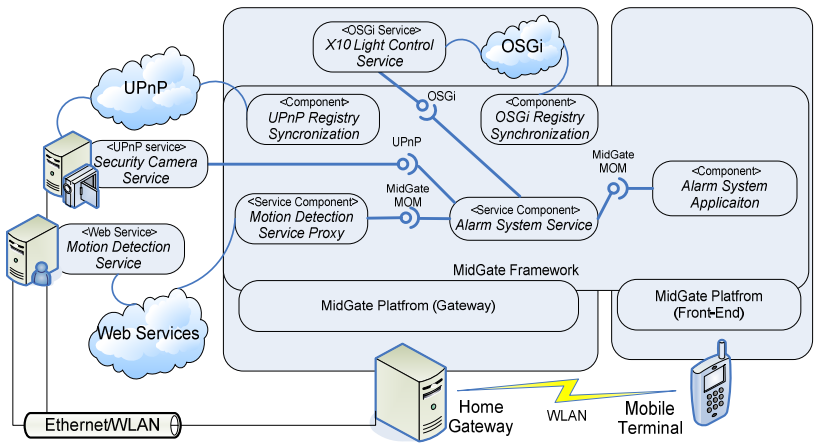


Fig. 8. Prototype system.

As illustrated in figure 8, the prototype system consisted of MidGate gateway host platform implementation deployed on a home gateway (a PC connected to home net-

work as well as to Internet) and MidGate front-end host implementation deployed on mobile terminal (Sony Ericsson P990i mobile phone). Reflecting to the MidGate system architecture presented in Figure 2, the prototype system consisted of the mobile terminal as a front-end host and two PCs as back-end host. The three service oriented software technologies used in the validation case were UPnP, Web Services (WS-Eventing [4]) and OSGi; The UPnP service was a security camera service (UPnP Digital Security Camera V1.0) providing still pictures and video feed from a web camera. The Web Service implemented following WS-Eventing specification was a motion detection service providing information from a motion detector device. The OSGi service was X10 light control service able to determine the lighting level and control the lights on request (ON/OFF). Additionally the message oriented middleware access through MidGate platform's internal message oriented middleware [9], [12] was used as a service binding and interaction technique within the MidGate Framework (MidGate MOM in figure 8.)

The system consisted of control application deployed on user mobile terminal as a MidGate component, and a composite service called Alarm System Service deployed as MidGate service component at the home gateway. As a composite service the Alarm System Service utilized the Security Camera Service via UPnP interface, X10 Light Control Service via OSGi interface and a Motion Detection Service via the a service's proxy component implemented as a MidGate service component providing a MidGate MOM based interface. Additionally the system included two MidGate components implementing automatic service registry synchronization between UPnP and OSGi service registries and the MidGate framework. The registry synchronization components listened for service registrations within their respective external service ecosystem and captured and registered necessary information for identifying and contacting the services towards the MidGate Framework. Accordingly, both two options for integrating external service ecosystems were applied in the prototype; service proxy and service registry synchronization.

The application scenario was successfully implemented and demonstrated functionally validating the capability of MidGate Framework to integrate services from multiple external service ecosystems into interoperable service composition.

6. Conclusion

A novel service component framework, called MidGate Framework, enabling interoperable registration, discovery, binding and interaction of modular software services implemented in multiple different service oriented software technologies, was presented in the paper. An overview of the MidGate Framework and related MidGate system architecture was presented to illustrate the main integration concepts for service oriented software technologies and the technological context of the work done. The service component structure and lifecycle actions within the service component framework were presented in detail to illustrate the reflection of collaborative service and component oriented design towards the different parts and their interrelations within a service component. The framework architecture designed based on separa-

tion of component and service lifecycle management was presented highlighting the main architectural design patterns applied.

As presented in the paper, the service component framework can be realized as a core part of a gateway based middleware service platform, called MidGate platform, providing a deployment environment for interoperable and modular service oriented applications in highly dynamic and technologically heterogeneous pervasive computing environments. As a main novelty of the work, the capability of the service component framework, to enable interoperable registration, discovery, binding and interaction of software services implemented in multiple different service oriented software technologies, was functionally validated and demonstrated via a laboratory prototype system overviewed in the paper.

References

1. Barbier, G.: Service Component Architecture Home. Web Page, URL: http://www.osoa.org/display/Main/Service+Component+Architecture+Home. (2007)
2. Bohn, H., Bobek, A., Golatowski, F.: SIRENA - Service Infrastructure for Real-Time Embedded Networked Devices: A Service Oriented Framework for Different Domains. In: International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies 2006. ICN/ICONS/MCL 2006. (2006)
3. Booth, D., Haas, H., McCabe, F.: Web Services Architecture. Web Page, URL: http://www.w3.org/TR/ws-arch/. (2004)
4. Box, D., & Caprera, L. F.: Web Services Eventing (WS-Eventing). Web Page, URL: http://www.w3.org/Submission/WS-Eventing/. (2006)
5. Erl, T.: SOA Principles of Service Design. Prentice Hall, Crawfordswille, Indiana, USA (2007)
6. Escoffier, C., Hall, R. S., Lalanda, P.: IPOJO: An Extensible Service-Oriented Component Framework. In: IEEE International Conference on Services Computing 2007. SCC 2007. ,pp.474-481, (2007)
7. OSGi Alliance: OSGi Service Platform Core Specification, Release 4, Version 4.1. 288p. (2007)
8. OSGi Alliance: OSGi Service Platform Service Compendium, Release 4, Version 4.1. 710p. (2007)
9. Paakkonen, P., Pakkala, D., Sihvonen, M.: An Optimized Message-Oriented Middleware Solution for Extending Enterprise Services to the Mobile Domain. In: Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services 2005. ICAS-ICNS 2005. (2005)
10. Pakkala, D., Koivukoski, A., Latvakoski, J.: MidGate: Middleware Platform for Service Gateway Based Distributed Systems. In: 11th International Conference on Parallel and Distributed Systems 2005. ICPADS'05, pp.682-688, (2005)
11. Pakkala, D., & Latvakoski, J.: Distributed Service Platform for Adaptive Mobile Services. International Journal of Pervasive Computing and Communications, 2**,** pp.175-187 (2006)
12. Pakkala, D., Pakkonen, P., Sihvonen, M.: A Generic Communication Middleware Architecture for Distributed Application and Service Messaging. In: Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services 2005. ICAS-ICNS 2005. (2005)

13. Pakkala, D., Perälä, J., Niemela, E.: A Component Model for Adaptive Middleware Services and Applications. In: 33rd EUROMICRO Conference on Software Engineering and Advanced Applications 2007. pp. 21-30 (2007)
14. Schmidt, D. C.: Middleware for Real-Time and Embedded Systems. Communications of the ACM, 45, pp.43-48 (2002)
15. UPnP Forum: Universal Plug and Play (UPnP) Forum Web Page. URL: http://www.upnp.org/ (2007)

Service Contract Compliance Management in Business Process Management

Marwane El Kharbili[1] and Elke Pulvermüller[2]
[1]ARIS Research, IDS Scheer AG. Altenkesselerstrasse 17 66115 Saarbrücken, Germany
marwane.elkharbili@ids-scheer.com
[2] Institute of Computer Science, University of Osnabrück. Albrechtstr. 28, 49076, Germany
elke.pulvermueller@informatik.uni-osnabrueck.de

Abstract. Compliance management is a critical concern for corporations, required to respect contracts. This concern **is** particularly relevant in the context of business process management (BPM) as this paradigm is getting adopted more widely for designing and building IT systems. Enforcing contractual compliance needs to be modeled at different levels of a BPM framework, which also includes the service layer. In this paper, we discuss requirements and methods for modeling contractual compliance for an SOA-supported BPM. We also show how business rule management integrated into an industry BPM tool allows modeling and processing functional and non-functional-property constraints which may be extracted from business process contracts. This work proposes a framework that responds to the requirements identified and proposes an architecture implementing it. Our approach is also illustrated by an example.

1 Introduction

Businesses are subject to regulations acting on a certain business domain, and to contracts signed with business partners. These businesses' value creating activities as well as intrinsic knowledge (e.g. organizational) are contained in the enterprise's business processes (BPs). Business contracts do not only contain the definition of stakeholders, transactions taking place, required services to be exchanged between stakeholders (functional aspects). Business contracts (BCs) place constraints and provide preferences for the latter (non-functional aspects) as well. The agreement on business process contracts (i.e. contracts on transactions carried out by BPs) must be followed-up by checking compliance of the transactions being actually carried out to BCs. Enterprise compliance management has been defined in [1] as the term referring to standards, methodologies, frameworks, and software used to ensure the company's observance of legal texts. Here, we understand business contracts as one kind of legal text. In the context of BPM, compliance management applies on business processes and the related resources like services, data and systems. As in [15], we regard services as abstract entities providing a functionality, being either web services, human resources or any appliance (e.g. airport ticket terminal). Modeling and checking compliance to business contracts, in the context of an SOA-enabled BPM platform, implies the need for modeling and enforcing these business contracts on the elements composing the SOA, i.e. on services. BPM takes a model-driven approach to

designing IT systems. The top layer (conceptual BP models layer) contains design time BP models written in notations (e.g. BPMN[1]). The underlying layer contains executable process models written in notations such as BPEL which can be run on execution engines and may require making calls to web services. These lie on the bottom layer which provides functionalities looked after and composed into executable processes. This implies that **different representation formalisms for business contracts are available on the different layers of a BPM framework, and that the latter need to be translated in representations that are adequate for being expressed on services**. There have to be transformation mechanisms to transport constraints extracted from business contracts between a BPM framework's layers. Provided that **a formalism for modeling business contracts is given, we assume that we can generate business rule descriptions of such a business contract**. In the context of an industrial BPM platform taking supporting this model-driven approach to BPM, we present a **method for managing service contracts in a BPM platform and showcase how using declarative business rule-based modeling of service contracts can be realized** in this platform.

In the following sections, we will further discuss the problem of ensuring contractual BP compliance for services and then explain our approach for this matter. Section 4 introduces a high-level architecture for implementing our approach and Section 5 uses an example to illustrate it. A related work analysis follows in Section 6 and we conclude and present future work in Section 7.

2 Contractual Service Level BP Compliance Management

In a BP oriented enterprise, service contracts need to be (i) formally modeled, (ii) checked (verified for formal correctness and validated for adequacy to incepted needs), (iii) deployed in the IT architecture carrying out business activities, (iv) enforced on several layers of the enterprise's business and IT systems, and finally (v) monitored in order to ensure that the eventual use made of these contract models is correct. Service Level Agreements (SLAs) are defined as a formal agreement between a service provider and a service consumer which describes functional and non-functional properties of the interaction of the two. By using SLAs, service providers and requestors, the interaction of which makes up business transactions, can agree on and check conditions of their collaboration.

We illustrate our ideas in the context of the ARIS architecture for BPM [15]. Although it is possible to model capabilities and data input/output of services in ARIS, the ARIS meta-model does not include modeling of non-functional properties for services. Business contracts (BC) specify (i) which sets of services are to be expected by one BP transaction partner from another, (ii) and under which conditions. These conditions are usually expressed as a set of constraints. One possible way of representing the constraints posed by business contracts on services is to use some formal logic, such as business rules. Business rules can be interpreted by agents for checking SLAs and the execution of business processes would be dependent on the

[1] http://www.bpmn.org/

decisions taken by these business rules. Furthermore, contracts may not simply express constraints on accepted behavior in a business transaction, decision processes may also be included. For example, a business contract may contain the following clause: *Trip booking service response time between 10:00 am and 16:00 am must be smaller than 0.8 s, else, an additional load balancing trip booking service must be made accessible in order to keep global response time of trip booking services under 0.8 s.* Such clauses are decisions. Some clauses my even contain complex decision processes. Modeling these complex decisions must be supported by a service-level contractual compliance framework.

In order to allow consistent SLA modeling and evaluation at each BP transaction partner, a commonly agreed on model for functional and non-functional properties for services is needed. Otherwise, SLA evaluations at one partner would not rely on the same informational basis as other partners, thus making these SLA evaluations irrelevant for BP transaction partners. The common information model (CIM[2]) and the OMG's UML profile for quality of service[3] (QoS) are two examples of such a common vocabulary. As contract clauses are written in natural language and maintained by human users, they may be subject to inconsistencies. Some contract clauses relevant for BPs and services may be in contradiction to others. A formal language for modeling SLAs making use of logic is one possible solution, since it allows reasoning on big amounts of SLA assertions and checks them for satisfiability. Moreover, additionally to SLA verification, SLAs must be validated for adequacy to business contract requirements. Typically, SLAs require metrics to be defined, which are used in checking if SLA entries are held. Metrics can be further processed in the form of key performance indicators (KPIs) which process SLA metrics and compute values making sense at a process level. The idea here is that SLA modeling and enforcement only makes sense if it supports BP contracts. Hence, an explicit link between individual BP contract clauses and supporting KPIs and SLAs is necessary. To this purpose, a formal representation of business contracts (such as in [16]) and a formal mapping of contract representations to SLA descriptions required. This additional layer of KPIs on top of metrics for SLAs allows real-time monitoring BP executions by building a KPI model and displaying computed results in a dashboard.

3 A Rule-Based Approach to SLA Modeling

SLAs may contain these distinct parts:

- An agreed on NFP-Model upon which service levels will be defined and calculated.
- A specification of calculation models and metrics used to asses service levels. These are called service level indicators (SLI).
- A specification of the defined service levels, with regard to computed SLI values (i.e. SLI quantification).

[2] Distributed Management Task Force CIM schema specification - version 2.19, July 2008: http://www.dmtf.org/standards/cim/.

[3] UML Profile for Quality of Service and Fault Tolerance - version 1.1, April 2008: http://www.omg.org/docs/formal/08-04-05.pdf.

– A definition of the agreed on service levels to be delivered by transaction partners. These are called service level objectives (SLO) and are defined based on SLI quantifications.

Apart from the first one, which is tackled by NFP-Models, the previous points could be modeled using business rules, by expressing constraints on relevant on the NFP-Model. For expressing SLIs, corresponding mathematical (logical and arithmetical) formula can be encoded directly in rules, and evaluated against run-time instances of the NFP-Model. In the same fashion, SLI quantifications can be expressed into business rules provided these service levels are pre-defined into the NFP-Model. This can be done by making business rules set values for service level attributes of instances of the NFP-Model. SLOs can also be modeled into business rules by extending the NFP-Model with concepts for modeling SLOs, and making business rules set attributes for these. The simplest way of seeing this is that each SLO concept instance linked to a service has a Boolean attribute *SLO.attained* which is set by the business rule to either *TRUE* or *FALSE*.

Additionally, some SLAs may include more complex elements, which are more decision processes than service level evaluations:

– A pre-defined compensation mechanism for failed transactions. A transaction fails when its SLOs are not successfully met. Mechanisms need to be defined in order to be able to react to failure situations. These mechanisms can be modeled as decisions processes. A example would be: *If SLO1 is not met for the encryption service then the service provider must send a URI to another encryption service within 3 minutes of failure notification. This new service shall not fail for at least the next 50 transactions*.

– Pre-defined penalties need to be defined in case no compensation mechanisms are defined, or in case compensation mechanisms fail themselves. An example: *If the encryption service calls fail for two different services consequently then a money penalty of 0,05 euro per second for the duration of out-of-service must be paid to the service consumer*.

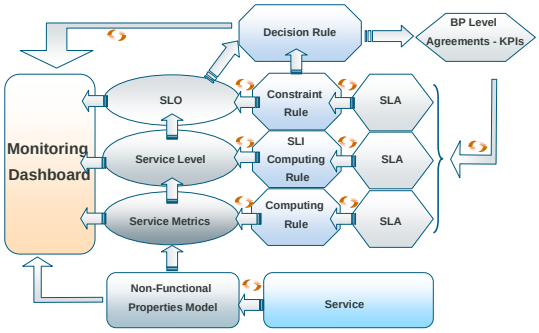


Fig. 1. An Approach for SLA-based Contractual Compliance

The previous points combine decisions with processes and can be modeled using rule flows. Rule flows are control flows which tasks are calls to individual business rules. Rule flows allow thus to model expected behavior by a service depending on service levels measured, which encapsulates more functionalities relevant for the service in one point. Hence, it becomes possible to change this behavior

independently from the functionality realized by the service itself. The same service interface can thus react differently depending on the transaction it is involved in and the SLAs it has defined with its transaction partners. Moreover, rule flows can trigger events or make direct calls to applications and systems. The latter are important in scenarios where compensations and penalties are taken in charge. However, the last two points are at the boundary between duties of a service level agreement and service management generally, and some web service specifications[4] already consider these aspects.

Our approach (Figure 1) seeks to extend the ARIS [13] tool with SLA management and builds on the requirements elicited in Section 2 and on the previous remarks from this section. We distinguish several layers for managing SLAs ranging from the NFP-Model to computing KPIs relevant for BP level agreements. Services are annotated with elements of the common NFP-Model. Services are also annotated with Rule-based. The latter are fed to a service level assessment agent responsible for deciding if individual SLOs for the service are met or not. the computed SLIs and SLOs can be further processed into process KPIs for process level agreements, aftre having been processed by an agent responsible for taking relevant actions depending on whether SLOs are met or not (e.g. compensation, penalties). Service metrics, SLIs, SLOs and KPIs can be monitored on a business process level compliance dashboard. We will see in an example later in this paper how SLA rules and NFP-Models are defined.

4 A Rule-Based Architecture for Service Contract Management

The following figure shows how our approach to modeling service contracts is built around the ARIS notation for BPM and the business rule modeling functionalities provided by ARIS. In ARIS, Business rules are modeled either as rule flows or as decision tables (called rule sheets). Rule flows are control flows of rule sheets and are used to model complex rules. Each rule flow or rule sheet uses a domain object model called vocabulary, in order to express constraints and decisions based on this object model. The language used in rule sheet decision tables to model decisions is a superset of OMG's object constraint language (OCL). Additionally, business rules can be used to generate web service descriptions that make it possible to invoke business rule logic as a web service. Business contract definitions are attached to functions in an event propagation chains (EPC[19]) process. The *FUNCTION* concept in the EPC notation is the same as the *ACTIVITY* concept in BPMN or the *TASK* in other BP modeling notations. As business level agreements are used to model business process relevant aspects of business contracts, each function of a BP may be annotated with one or several BLAs (blue object in Figure 2, upper part). In ARIS, a function can be assigned another EPC process. In our case the function *Customer Order Processing* is assigned another EPC process, of which we can observe one part on Figure 2 (middle-right part, the *plan production* function). We can see that one function of the *Customer Order Processing* EPC is annotated by (i) a BLA (blue 4-round-cornered shape), (ii) an SLA implementing this BLA (5-

[4] http://www.ibm.com/developerworks/library/specification/ws-tx/

cornered shape; note that the ARIS meta-model allows using several SLAs to model a BLA, (iii) an NFP-Model, (iv) an SLA rule implementing the SLA, (v) and finally a KPI object (brown 4-cornered shape). The BLA is inherited from the *Customer Order Processing* function and the SLA makes use of an NFP-Model modeled as a UML class diagram. Ultimately, the SLA rule expresses the *Schedule Calculation SLA* on this NFP-Model.

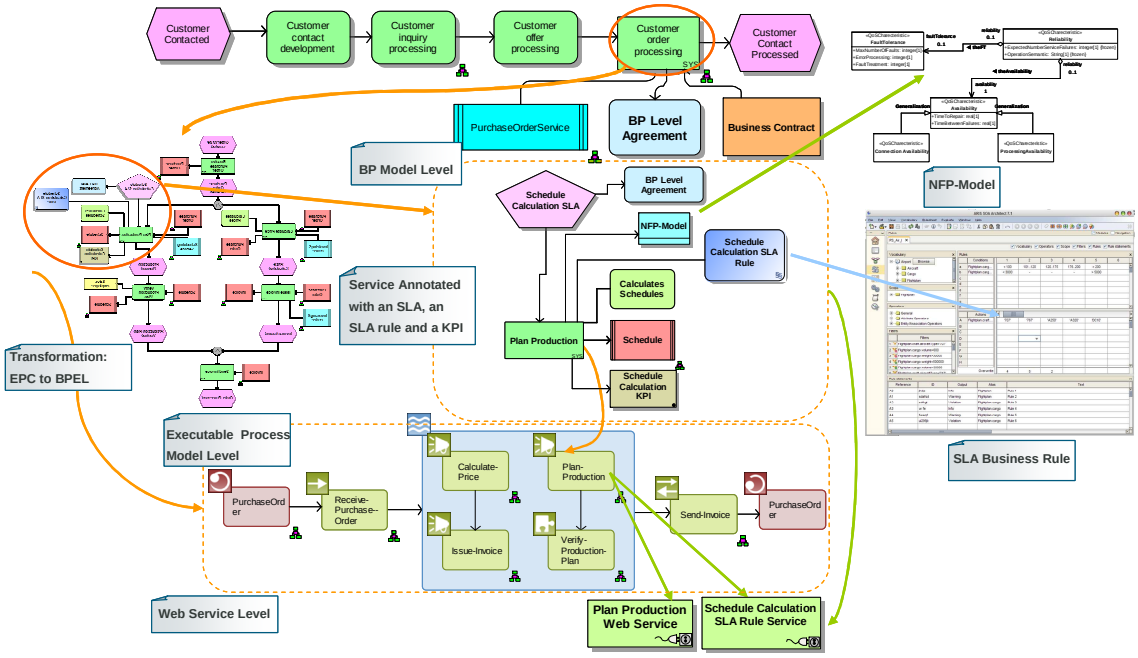


Fig. 2. model-driven architecture for service level management in the ARIS framework

The *Customer Order Processing* function is automatically transformed into a BPEL process (makes use of the model transformation introduced in [15] On Figure 2, we can see that the BPEL invoke activity *Plan Production* maps to the *plan production* EPC function on the overlaying layer. This invoke activity calls two web services in the following order: the *Plan Production* WS which realizes the *plan production* functionality followed by the *Schedule Calculation SLA Rule Service* WS which realizes the *Schedule Calculation SLA Rule*. This way, decisions and/or SLIs and SLOs computed by the *Schedule Calculation SLA Rule Service* are brought back to the process execution layer. To further process these results, the KPI object linked to the *plan production* EPC function must also be modeled as a business rules and made available on the service layer as a WS.

This architecture allows propagating decisions made by SLA rules by making the latter set values of instances of the NFP-Model linked to services. However, if business rules need to trigger additional activities depending on the decisions made at run-time, business rules should not simply be made available as WS, they should be taken into account by the BPEL transformation in the same fashion as EPC functions

are, and extend the generated BPEL processes to this purpose. The framework also needs to be extended in order to transform rule flows into BPEL and link BPEL invokes to Web Services generated from the rule sheets composing the rule flow. Our approach requires no specialized agents for enforcing SLAs and no special protocols necessary, since our work only considers service contract checking at design-time, based on the SLAs attached to BP functions, and verified on the services which can be assigned to these functions.

5 Example

To illustrate the concepts discussed in Sections 2 and 3 and explained in Section 4, we will use a small example of modeling SLAs for business processes. Be a business contract made between two partners A and B specifying that: The service provider of the trip booking service is *obliged to provide services with durations between each two service break-downs of more that the minimum fault-free time windows allowed by the respective subsidiary of partner A*. Additionally, partner B *must provide error-detection functionalities for its services and no more than 1 unrecovered failure every 15 failures is allowed*. The number of actual service failures must be at best equal to 1/20th of the necessary time to get a broken service back up and running.

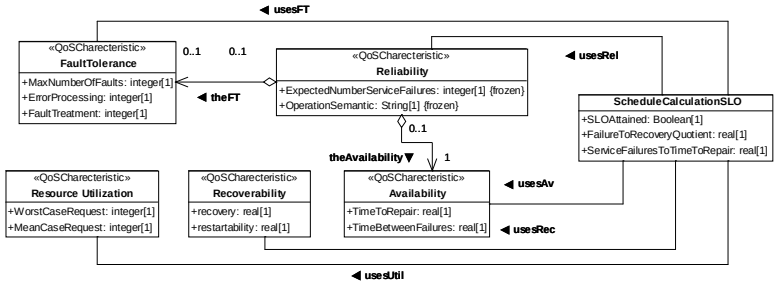


Fig. 3. Subset of the OMG's UML profile for QoS and Fault Tolerance – NFP-Model example

In order to model this clause of the business contract, we use the NFP-Model in Figure 3 which implements a subset of the OMG's UML profile for Quality of Service (QoS) and Fault Tolerance (for reliability, fault tolerance, availability, recoverability and resource utilization) and augment it with an SLO class for service level assessment. We model the following SLA constraints and the following KPIs for BLA elicitation:

- **SLI0: MaxNumberOfFaults.** attribute of the FaultTolerance QoSCharacteristic. For the portuguese subsidiary of partner A, the minimum duration of fault-free windows is of 200 seconds.
- **SLI1: FailureToRecoveryQuotient.** This SLI is computed as the ratio of the ExpectedNumberOfServiceFailures attribute of the Reliability QoSCharacteristic (See previous figure) to the Recovery attribute of the recoverability QoSCharacteristic.

– **SLI2: ServiceFailuresToTimeToRepair.** This SLI is calculated as the ratio of the ExpectedNumberServiceFailures attribute of the Reliability QoSCharacteristic to the TimeToRepair attribute of the Availability QoSCharacteristic.
– **SLO1:** If ((SLI0 < 200) & (SLI1 < 1/15) & (SLI2 < 1/20)) Then SLO1 is achieved.

Finally, we model these SLA constraints in the business rule shown in the following figure. Service level assessment is then conducted as explained in the previous section.

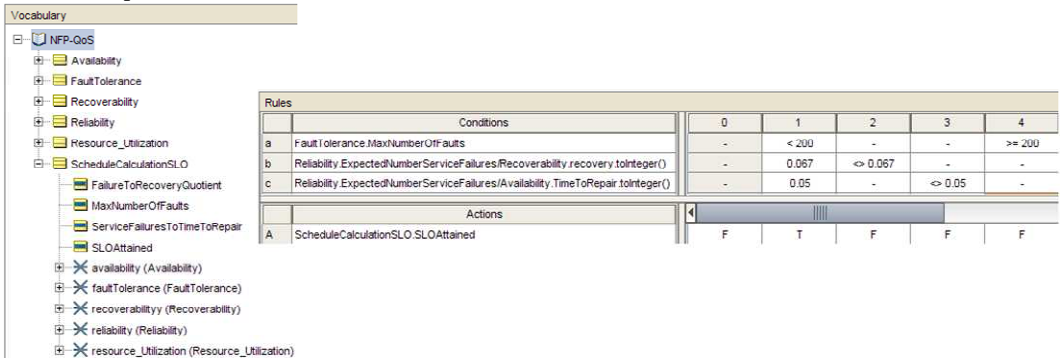


Fig. 4. Business rule for the example SLA

6 Related Work

Service level agreements are often cited in connection with service level management such as in [6] and most of the time, SLA frameworks are designed to support a full SLA management lifecycle [7]. Such lifecycles include design, negotiation, implementation and monitoring aspects. Apart from negotiation, these aspects are also taken into consideration by our approach, with the difference that this is done as part of an industry-accepted BPM framework, and only considers design-time aspects. Several SLA languages have been specified, such as the Web Service Offering Language (WSOL)[12], the Web Service Management Language (WSML) as part of the Web Service Management Network (WSMN)[11] specification, and SLAng[9]. SLAs are also often modeled using quality of service (QoS) descriptions, such as the one defined in [8]. The latest standard in this series is WS-Agreement (previously WSLA) [10]. Each of these languages proposes it own structure of meta-information for SLAs and nun-functional or QoS model. Although these languages could cover most needs of a solely service-centered framework, none of these languages has the formal expressiveness of business rules. Such standards do not provide linking to BPM layers. A very interesting approach is the work by Paschke [14] on the RBSLA (Rule Based SLA) solution which is similar to ours in that it also relies on the definition of business rules, and makes use of RuleML[5] to model SLAs. As Paschke

[5] http://www.ruleml.org/

underlines it in his work, XML-based SLA languages (e.g. WSOL, SLAng, etc.) require an interpreter and are limited to the expression of Boolean logic. These cannot make use of more powerful logic constructs such as variables or rule chaining, such as it is possible in our approach which makes use of a business rule engine for this. But again, RBSLA misses on integrating SLA management with a BPM framework.In [21], a general purpose non-functional properties language called Process^{NFL} is proposed which provides the possibility to express correlations and conflicts between NFPs as well as compositional aspects of NFPs. However, ProcessNFL has no formal semantics. An Additional QoS language is QuO[20], which according to [22] is a contract-based approach, that specifies actions to take in case QoS constraints are not met. However, QuO mixes declarative specifications of QoS with implementation specification which makes it quite complex.

7 Conclusion and Future Work

This work is a first direction towards service level enforcement of business contracts, in the scope of a BPM framework. We have introduced the scenario of BP contractual compliance and motivated requirements on a framework for this. We then proposed an approach for modeling service level agreements relying on business rules. In the context of the ARIS platform for BPM, an architectural approach for rule-based modeling of SLAs was given. We identify four main immediate directions of future work. First of all, a formalization of business contracts that is adequate for extracting SLAs should be developed and automatic transformations from these business contracts into BLAs and further into SLAs be designed. In [23], work has been conducted on modeling dimensions of BP quality, which could be used to model BLAs, and [24] focuses on quality of composite services, defining a contract model based on these. Also, modeling complex SLAs which involve complex decision making such as compensation, implemented by generating events or triggering external processes and are not limited to expressing constraints needs to be supported. Moreover, more work needs to be done in formalizing KPI modeling using business rules, in order to use this for BLA monitoring. Finally, we will seek to design a common information model for non-functional properties for the ARIS platform for service management. Making this common NFP-Model available to all services will enhance interoperability between services.

References

1. El Kharbili, M.; Stein, S.; Markovic, I. & Pulvermüller, E. Towards a Framework for Semantic Business Process Compliance Management. In Proceedings of the workshop on Governance, Risk and Compliance for Information Systems, June 2008 (pp. 1-15).
3. E. Pulvermüller, A. Ludwig, R. Belter, D. Zyskowski, U. Heindl, X. Nguyen, A Capability Oriented Management Approach for Business Integration. In 1st International Conference on Business Process and Service Computing (BPSC'07). Leipzig, Germany, 2007.

4. Dan, A., Ludwig, H., Pacifici, G. Web Service Differentiation with Service Level Agreements. 01.05.2003. http://www.ibm.com/developerworks/library/ws-slafram/.
5. Pandit, B., Popescu, V., Smith, V. Service Modeling Language, version 1.1. W3C working draft 12 September 2008. retrieved on 15.09.2008 from: http://www.w3.org/TR/sml/.
6. Sturm, R. Morris, W. Foundations of Service Level Management. Sams, 2000.
7. Lee, J.J., Ben Natan, R. Integrating Service Level Agreements: Optimizing your OSS for SLA delivery. Wiley, 2002.
8. Lee, K., Jeon, J., Lee, W. QoS for Web Srevices: requirements and Possible Approaches, W3C Working Group Note, 25.11.2003. Retrieved on 11.09.2008 from: http://www.w3c.or.kr/kr-office/TR/2003/NOTE-ws-qos-20031125/.
9. Lamanna, D. D., Skene, J., Emmerich, W. SLAng: A Language for defining Service Level Agreements. In proceedings of the 9th IEEE workshop on Future Trends in Distributed Computing Systems, 2003.
10. Open Grid Forum, WS-Agreement Specification. March 2007. Retrieved on 11.09.2008 from: http://www.ogf.org/documents/GFD.107.pdf.
11. Vijay Machiraju, Akhil Sahai, Aad van Moorsel. Web Services Management Network: An Overlay Network for Federated Service Management. HP Labs. Retrieved on 11.09.2008 from: http://www.hpl.hp.com/techreports/2002/HPL-2002-234.html.
12. Tosic, V., Pagurek, B., Esfandiari, B., Patel, K., Ma, W. Web Service Offerings Language (WSOL) and Web Service Composition Management (WSCM). In proceedings of the Object-Oriented Web Services Workshop (OOPSLA) 2002.
13. Scheer, A.-W. ARIS, vom Geschäftsprozess zum Anwendungssystem. Springer, 2002.
14. Paschke, A.: Rule-based Knowledge Representation for Service Level Agreements. Doctoral Symposium of MATES'06 (MATES'06), Germany, 2006.
15. Stein, S.; Lauer, J. & Ivanov, K. ARIS Method Extension for Business-Driven SOA Wirtschaftsinformatik, 2008, 50.
16. Governatori, G. & Z, M. Dealing with contract violations: formalism and domain specific language. Proceedings of the Conference on Enterprise Computing EDOC 2005. IEEE Press., 2005, 46–57.
17. Miller, J. & Mukerji, J. MDA Guide. Object Management Group (OMG), 2003.
18. Wagner, G.; Giurca, A. & Lukichev, S. A Usable Interchange Format for Rich Syntax Rules Integrating OCL, RuleML and SWRL. Proceedings of the World Wide Web Conference, 2006.
19. Keller, G.; Nüttgens, M. & Scheer, A. W. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Universität des Saarlandes, 1992.
20. Loyall, J., Bakken, D., Schantz, R., Zinky, J., Karr, D., and Vanegas, R. Qos aspect languages and their runtime interactions. In languages, compilers, and run-time systems for scalable computers. LNCS 1511. Springer-Verlag.
21. Rosa, N. S. et al.. Process NFL: A language for describing non-functional properties. In Proceedings of the 35th Annual Hawaii International Conference (HICSS), March 29, 2005,Hawaii, *USA*, pp. 3676–3685.
22. Toma, I., Foxvog, D. Non-functional properties in web services. WSMO Working Draft – October 25, 2006. retrieved on 09.09.2008 from: http://www.wsmo.org/TR/d28/d28.4/v0.1/.
23. Heravizadeh, M., Mendling, J., Rosemann, M. Dimensions of Business Processes Quality (QoBP). In: Proceedings of the 6th International Conference on Business Process Management Workshops, Milan, Italy, 1 September 2008.
24. Comuzzi, M., Fugini, M., Modafferi, S.. Quality Contracts for Cooperative Services and Associated Resources. In Proceedings of the 2nd International Workshop on Collaborative Business Processes, Milan, Italy. 2008.

An Architecture for Autonomic Web Service Process Planning

Colm Moore and Ming Xue Wang and Claus Pahl

Dublin City University, School of Computing, Dublin 9, Ireland
christopher.moore4@mail.dcu.ie, [mwang|cpahl]@computing.dcu.ie

Abstract. Web service composition is a technology that has received considerable attention in the last number of years. Languages and tools to aid in the process of creating composite web services have been received specific attention. Web service composition is the process of linking single web services together in order to accomplish more complex tasks. One area of web service composition that has not received as much attention is the area of dynamic error handling and re-planning, enabling autonomic composition. Given a repository of service descriptions and a task to complete, it is possible for AI planners to automatically create a plan that will achieve this goal. If however a service in the plan is unavailable or erroneous the plan will fail. Motivated by this problem, this paper suggests autonomous re-planning as a means to overcome dynamic problems. Our solution involves automatically recovering from faults and creating a context-dependent alternate plan.

1 Introduction

The Semantic Web is an emerging technology that creates some opportunities in the field of Web services. Automatic composition of semantically described services is an example. Sequencing services together to accomplish more complex tasks can create difficulties when automated at runtime. AI planners can provide a solution in the form of a plan (often a sequence of Web services required to solve the problem at hand). Once these plans have been made, composite web services can be generated and invoked. However, what will happen if a service becomes unavailable or is not functioning properly? As a solution to this problem, we suggest an execution, monitoring and re-planning architecture.

A second component is the service process generation, which creates an executable process from an abstract plan. The component must convert the plan into an executable process. Using this information, a composite Web service is constructed that can communicate with the services in the plan and execute them in order. This service process needs to be deployed on a Web server and then invoked by the execution component.

If an expected result is returned it means that a Web service has executed without problems. If, however, the fault handling mechanisms indicate an error has occurred, other action must be taken. The third and central component is a monitoring and analysis that detects execution problems and analyses possible

remedies. Re-planning results in a new plan that contains alternate Web services that can also accomplish the same task. It is necessary for our program to get an alternate plan from the planner and start the execution process again.

A number of papers discuss the automation of service composition. McIlraith and Son [6] use the AI planner Golog [5]. Golog is a logic programming language based on the Situation Calculus, build on top of Prolog. Other planners, like hierarchical task network planners such as SHOP2, are based on the situation calculus. When composing Web services, high level generic planning templates (subplans) and complex goal can be represented by Golog. While these approaches can provide acceptable plans, this technology needs to be adapted to a dynamic environment. We have already identified two components of an architecture – process conversion and process monitoring and analysis – that can accomplish this integration.

An outline of the entire autonomic planning process follows in the Section 2 and introduce service composition, planners and process execution. Section 3 details the autonomous process planning. In Section 4, we introduce the overall system architecture, which is subsequently discussed in detail in terms of plan execution (Section 5) and monitoring and replanning (Section 6). We end with a discussion and some conclusions.

2 Dynamic Composition and Planning

In order to derive from Web service description an executable composite Web service automatically at runtime, a number of steps and transitions are required. Two central activities are plan generation based on abstract goals and service descriptions and plan conversion for execution through an execution engine.

2.1 Service Composition and Process Planning

A crucial step is to create a plan from service descriptions. AI planners are tools that are used to determine a plan, which is composed of a series of actions, an initial state, a goal state and a set of possible actions. SHOP2 is a Hierarchical Task Network (HTN) planner. In HTN planners, each state of the world is represented as a set of atoms with actions corresponding to deterministic state changes [7]. The planning domain is represented by operators (tasks) and methods. The methods decompose a set of complex tasks into subtasks. The plan is a sequence of these tasks. In the case of Web service composition, services are represented as operations. Inputs and outputs of services are represented as preconditions and postconditions joined with other semantic information. The plan is a sequence to execute the services in order to achieve the predefined goal.

The first requirement is to define a goal or overall task that is required. The goal is the desired outcome from the system once it has finished executing. This goal will usually require a series of Web services executions and, most likely, a number of message transactions. For a simple example, the purchase of a book would require first the lookup of stock to make sure the book is

available and then the credit transaction. The Web service information gathered will be automatically translated into an AI planner interpretable language from a knowledge-based language such as OWL-S or WSMO [1]. The converted file then contains service information (input/output, pre-/postconditions of operations). The plan is initially not in an executable format. WS-BPEL is a language that allows for the composition and invocation of Web services. WS-BPEL engines are composite service executors. WS-BPEL connects to WSDL directly and provides error handing mechanisms.

Problems can occur during the execution of these processes. Web services are often not reliable, which affects both the composition and execution activities. Web service can become unavailable for many reasons. If this happens between discovery and invocation, the goal becomes unachievable. Using error handling and re-planning it is possible to recover from problems.

2.2 Web Service Composition

Web services are platform-independent Internet-accessible software components [10]. WSDL files describe the service and how to connect and interact with it. Web service composition is the linking of Web services to perform some new complex task. WS-BPEL (Business Process Execution Language) is an orchestration language used to define business processes based on Web services. It controls message passing and execution of the process. The message handling in WS-BPEL refers to WSDL to define how the incoming and outgoing messages are handled. WS-BPEL defines how the services can be scheduled and organized into an executable process that provides an integrated service [8]. WSDL files are defined as "partnerLinks" where their role in relation to the WS-BPEL file is determined.

3 Autonomous Service Process Planning

The purpose of this investigation is to address dynamic re-planning in Web service composition. This involves using the outlined technologies to actually build a system dynamically and automatically. This system must be capable of creating plans, converting them to a usable language and then executing them. In addition, the system must detect and handle errors from faulty Web services and then automatically create a new alternate plan. The context of the system determines the quality and consequence of errors.

3.1 Service Description

We use a book search feature as our running example. Four OWL-S files describing four basic services define the service repository used here. There is service to find information on a book given a title, two alternative services that find the price of the book from an ISBN number and a service that converts the price from one currency to another. The goal of the problem is to get a price for a

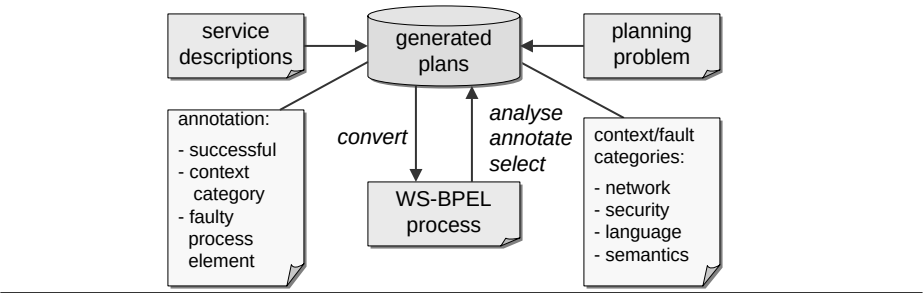


Fig. 1. Information Architecture

book in a given currency from the title of the book. We assume the four services as the result of a discovery activity.

```
<rdf:RDF xml:base="BookFinder.owl">
  <owl:Ontology rdf:about=""> ... </owl:Ontology>
  <!-- Service, Profile, and Process descriptions -->
   <process:AtomicProcess rdf:ID="BookFinderProcess">
    <service:describes rdf:resource="#BookFinder"/>
    <process:hasInput rdf:resource="#BookName"/>
    <process:hasOutput rdf:resource="#BookInfo"/>
  </process:AtomicProcess>
  <process:Input rdf:ID="BookName">
    <process:parameterType rdf:datatype="..">string</process:parameterType>
    <rdfs:label>Book Name</rdfs:label>
  </process:Input>
  <process:Output rdf:ID="BookInfo">
    <process:parameterType rdf:datatype="..">Book</process:parameterType>
    <rdfs:label>Book Info</rdfs:label>
  </process:Output>
  <!-- Grounding description -->
</rdf:RDF>
```

3.2 Planning

A planner generates an execution plan based on a given planning domain and planning problem, see Fig. 1. In SHOP2, the planning domain is established by a set of operators and methods. The input and output of the services are represented as preconditions and postconditions, respectively. For example, the book lookup service requires a book title to function; for the operator this would have a precondition that requires a BookName element to be accessible.

SHOP2 operator definitions consist of different parts: preconditions, which guards the operator execution. A delete list for negative postconditions and a add list for positive postconditions.

```
(:operator (!BookFinderService)
```

```
( (BookName ?bookName) )   ; preconditions
()                         ; negative postconditions
( (BookInfo bookInfo) ) )  ; positive postconditions
```

This SHOP2 interpretable code shows the BookFinder operator. The precondition is that there is a book name available. There is nothing in its delete list and its add list contains BookInfo (information about the book). Once the operation is executed, the process will have the variable BookInfo available.

In addition to operators, planning methods define how composite tasks are decomposed. A simple method includes a precondition and the subtasks that need to be accomplished in order to accomplish the composite task.

```
(:method (GetBookPrice)
 ( (BookInfo ?bookInfo) (Currency ?currency) )
 ( (!BookFinderService) (!AmazonPriceService) (!CurrencyConverterService) )
```

If preconditions are satisfied, the method decomposes GetBookPrice into subtasks, composed of BookFinderService, AmazonPriceService and CurrencyConverterService. A second GetBookPrice method has a different ShopPriceService.

3.3 Goals and Plan Creation

In addition to the operator and method input files, a planning problem file is created that represents the goal of the plan. When the Java version of SHOP2 executes, it takes the two files and converts them to Java, which can subsequently be executed to attain the plan. As there are alternate services available that can implement identical functionality as defined in the methods, there can be multiple plans. In the example GetBookPrice, when book name and a desired currency format is available in the initial state, SHOP2 returns two separate and both equally valid plans for the book price conversion goal:

```
Plan 1: BookFinderService; AmazonPriceService; CurrencyConverterService;
Plan 2: BookFinderService; BNPriceService; CurrencyConverterService;
```

SHOP2 can create an indexed list of plans. Multiple plan generation is a central feature since it allows different alternative plans to be executed in case of failure without the need to re-start the planning itself. Plan 2 above is such an alternative. Differences between plans can be noted and future selection can be based on this. We create an index to a plan repository to enable efficient access.

4 An Execution, Monitoring and Planning Architecture

A monitoring system with two components is the backbone, see Fig. 2:

– The first component is an autonomous plan execution component. Its aims are: conversion of abstract plans into executable service processes, pre-execution preparation of the execution environment including service description and deployment files, but also context fault-handling determination in addition to plan conversion, execution of the process using a service process engine.

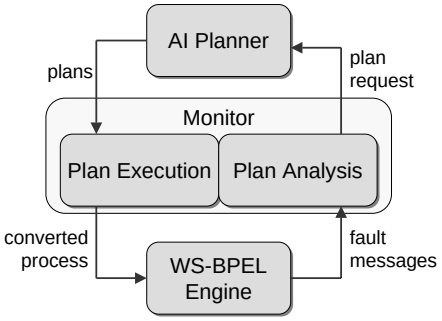


Fig. 2. System Architecture

– The second component is the context-dependent replanning component. Its objectives are: monitoring of process execution and fault capture, analysis of faults that have occurred during execution and determination of remedies (includes use of alternate existing plans or restart of planning process).

The necessary infrastructure consists of an execution engine at the core. The WS-BPEL execution engine that is used in this project is ActiveBPEL. It is an open source project written in Java. In terms of choice, the two most popular open source engines are Apaches ODE and ActiveBPEL. In terms of performance, the Apache engine has the advantage. ActiveBPEL however, provides excellent support for its engine, including many online guides and an actively monitored forum. In terms of the infrastructure, additionally Ant scripts add files to ActiveBPEL deployment folder. To simplify the integration of the planner into the architecture, the use of the Java version of SHOP2 called JSHOP2 is used instead of the Lisp version. The planner creates Java files to represent the problem/goal and the service description data.

5 Autonomous Plan Execution

5.1 Plan Conversion

Plan conversion involves two activities: conversion of the SHOP2 generated plan to a WS-BPEL representation and provision of input WSDL services and WS-BPEL deployment files for the BPEL engine. As part of the actual conversion of the plan into an executable process in WS-BPEL, a number of files need to be created. These are the WSDL files of the Web services that the plan requests to be invoked, the WSDL file of the generated WS-BPEL process and a number of deployment files, which are created by the WS-BPEL deployment tool.

5.2 Plan Execution

Plan execution – the second subcomponent – involves two activities: execution of WS-BPEL code and input OWL-S to XML parsing, which is done on the fly. As

the sample data originates from a number of OWL-S files, it is necessary to search through these to determine the location of the WSDL files which are needed for the WS-BPEL process, as WS-BPEL does not refer to OWL-S directly. This is done through XML parsing. Once the location is found, the WSDL file is analyzed and relevant information is selected. Information such as the how to connect, what message formats are needed, the names of services and others details are vital for the correct invocation of a service by the WS-BPEL process.

Creating the WS-BPEL file and its "partnerlink" WSDL file is done automatically. WS-BPEL files contain a number of sections which each have a particular role, sections such as partnerLinks, variables, faultHandlers and flow. These sections are made up individually and added to the file as they are required. Each section containing a template for standard layout in a section with relevant information simple is inserted as required. Information about Web service invocations is taken from the relevant WSDL file.

Here is a brief structural outline of the WS-BPEL specification:

```
<process>
   <partnerLinks>
      <partnerLink   name="BookFinder"
                     partnerLinkType="print:FinderLink"
                     partnerRole="BookFinderProcess"/>      ...
   </partnerLinks>
   <variables>
      <variable      name="BookName" ...  />      ...
   </variables>
   <flow>
      <invoke> partnerLink="BookFinder"
               operation="find" inputVariable="BookName" </invoke>
      <invoke> partnerLink="BookPriceCalc" ... </invoke>
      <invoke> partnerLink="PriceConvert" ...  </invoke>
   </flow>
<process>
```

Once WS-BPEL is created, it is deployed. Using the ActiveBPEL execution engine, deployment involves using Apache's Ant. This causes the files to be added to the ActiveBPEL's deployment folder and then deployed once it is noticed.

The deployed WS-BPEL service can be invoked from a manager component. Values are passed to the service; in our example the values would be the name of the book and information about the currencies needed. Once this invocation is made, the WS-BPEL process begins to execute its Web service references.

6 Monitoring and Context-dependent Replanning

6.1 Fault Handling

A vital element of WS-BPEL is fault handling. This is important due to the possibility if failure, but essential to our context to achieve autonomy. Fault

handlers can be defined in WS-BPEL to handle the exceptions thrown when a process is executing. Adding handlers to the invocations of Web services allows us to catch a fault when it arises. When a fault occurs and fault handlers have been defined, we use handlers to determine remedies in order to achieve the overall execution goal. Technically, a reply message indicating the fault is send by the handler (part of the execution engine) to the monitor (a separate component).

6.2 Context

In order to structure the failure handling aspect, possible failures are organised into context categories. The context notion refers here to execution environment factors that might impact the execution (and result in failure).

Context constraint violations need to be analysed and solutions determined. We distinguish for this implementation a number of (not necessarily exhaustive) context constraint violation categories:

- non-responsiveness of services: the service invoked does not respond
- security: a desired level of security cannot be achieved
- performance: the requested service cannot deliver efficiently enough

6.3 Analysis

The analysis component determines the actions to be taken from a failure in order to achieve the overall goal. It carries out the following steps:

- analysis of context constraint violation: an initial configuration can indicate whether violations of constraints are acceptable,
- a short planning cycle is necessary if violations are not acceptable: the analysis component detects previously generated plans (using the plan index) that can be tried as a remedy.
- a full planning cycle is necessary if violations are not acceptable and previously generated plans are not suitable (or not available): an invocation of the planner with the original goal is necessary.

Clearly crucial here is the decision whether a a time-consuming replanning (and possible service discovery) is necessary or whether an existing alternative plan can be used. This decision is context- and state-dependent. We annotate the indexed plan repository as follows: successful plan completion rate (probability of successful execution), fault type and associated context category, fault-generating plan/process elements. The plan annotation actually allows sets of fault type / process elements as a plan execution can cause different faults.

By distinguishing short- and full-cycle replanning, we achieve an improvement of planning performance; repeated generation of unsuccessful plans is avoided. The plan repository is updated (annotated unsuccessful ones) In the future, we aim to improve the annotation and analysis of unsuccessful plans. We plan to implement a learning technique that reliably allows to determine plans with a high degree of success from a plan repository. Clustering of faults/context categories and fault-causing elements is at the core of this endeavour. Our observation so far is that the success probability depends on the context category.

6.4 Implementation

Our WS-BPEL process has a number of fault handlers defined – corresponding to the context categories under consideration. In the case of an inaccessible service for instance, an error will occur. At this point the fault handlers take over. An automatic reply is sent to the monitor. If this message is a fault message and it indicates a non-responsive service the analysis component is called. It knows the plans that have already been produced and which of those have been (unsuccessfully) executed. It takes the next plan from the AI planner.

7 Discussion and Related Work

The solution that we implemented through our prototype indicates that an autonomic composition approach is feasible. Some concerns have, however, arisen.

A challenge that we encountered was the correctness of the conversion of a Web service composition plan into an actual working service process. Plans are abstract instructions, whereas WS-BPEL is executable process language with binding and deployment information. Information gathered, interpreted and converted to the correct format. This would include creating the WSDL files (from an OWL file) and extracting the data from these files to define a process that complies with the plan specification.

We have already discussed that performance is crucial and that we have provided a solution that targets plan reuse without replanning whenever possible. Improvements in this respects are, however, still possible. We mentioned an intelligent, context-dependent plan selection feature as a promising direction. We have focussed on communications-specific fault categories in our context definition. A range of other context aspects such as language, semantic context, a full range of quality criteria, etc. can be considered.

Many planning tools have been integrated into autonomic composition architectures. In [6], Golog is used as the planning component. In [7], with SHOP2 the same planner that we used is proposed based on OWL-S semantic Web service descriptions. [9] applied planning using a model checking approach. The plan generation is done by exploring the state space of the semantic model. In a recent hybrid AI planner [3], different planning techniques are combined. The major focus of these activities is discovery and service composition. However, they are lack fault-tolerance, which in distributed service infrastructures is a necessity for reliable implementations.

Many researches are looking into self-healing mechanisms [4] for service composition to achieve dependable systems. The self-healing approach focuses on monitoring and recovery activities for overcoming faulty behaviours of service oriented system. In [2], a self-healing composition strategy is defined, which includes assertion-based monitoring, event-based monitoring, history-based monitoring, recovery through a retry-failure service, recovery through a substitute-failure service, and recovery by restructuring plans. [11] presents an enhanced BPEL engine for self-healing. The engine is extended by planning, monitoring,

diagnosis and recovery modules. However, none of these activities provides a complete architecture solution for autonomic service composition.

8 Conclusions

In this paper, the problem of dynamic Web service composition and execution failure and error handling and re-planning has been addressed. The causes of this problem and the effects have been discussed. An architecture for autonomic, i.e. dynamic and automated service composition has been discussed.

One of the crucial characteristics of autonomic composition is a self-healing ability of the dynamically deployed composition system. It needs to deal with execution faults of a very different nature. We have proposed a context-based fault handling strategies that efficiently determines remedies in terms of reuse of plans or AI-based replanning and subsequent plan conversion. As indicated, our aim is to extend the current system by considering more context categories and to make the decision processes more efficient and reliable.

References

1. OWL-S Coalition. *OWL-S 1.1.* http://www.daml.org/services/owl-s/1.1, 2003.
2. S. Guinea. Self-healing web service compositions. *27th International Conference on Software Engineering*, 2005.
3. M. Klusch and A. Gerber. Semantic web service composition planning with owls-xplan. *1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, 2005.
4. P. Koopman. Elements of the self-healing system problem space. *Workshop on Software Architectures for Dependable Systems*, 2003.
5. H.J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R.B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–83, 1997.
6. S. McIlraith and T. Son. Adapting golog for composition of semantic web services. *Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, pages 482–493, 2002.
7. D. Nau, T. C. Au, O. Ilghami, U. Kuter, W. J. Murdock, D. Wu, and F. Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20:379–404, December 2003.
8. L. Padgham and W. Liu. Internet collaboration and service composition as a loose form of teamwork. *Journal of Network and Computer Applications*, 30(3):1116–1135, 2007.
9. M. Pistore, P. Bertoli, E. Cusenza, A. Marconi, and P. Traverso. Ws-gen: A tool for the automated composition of semantic web services. *3rd International Semantic Web Conference*, 2004.
10. B. Srivastava and J. Koehler. Web service composition - current solutions and open problems. *ICAPS'2003 Workshop on Planning for Web Services*, 2003.
11. S. Subramanian. On the enhancement of bpel engines for self-healing composite web services. *IEEE Symp. on Applications and the Internet*, pages 33–39, 2008.

Towards Service Architectures in Service-oriented Computing

Matti Mäki and Daniel Pakkala

VTT Technical Research Centre of Finland
Kaitoväylä 1, Oulu, P.O. Box 1100, FI-90571
{Matti.Maki, Daniel.Pakkala}@vtt.fi

Abstract. Service-oriented architectures (SOA) are nowadays a widely promoted field of study in service-oriented computing (SOC) but unfortunately often discussed only in the light of enterprise IT solutions and the Web services technologies. By diving into the technical fundamentals of SOA we found a more general concept of service architectures; a concept that might have much more application possibilities than its near relative, SOA. This paper presents a simple but feasible model for service architectures, based on the existing state-of-the-art research of SOC. Feasibility of some existing service platforms as service architecture realizations is evaluated against the model. The simple model provides a good starting point for researching and developing more sophisticated service architectures and a set of criteria for evaluating service platforms.

1 Introduction

Service-oriented architecture (SOA) has gained much attention during the recent years in software and services research. This trend seems to be mainly driven by enterprise businesses on their striving for more flexibility and interoperability in their information systems. The idea of service-orientation would be applicable in any context but the currently domimant SOA-approach is mostly defined in terms of enterprise IT systems and Web services, for example in [1], [2] and [3]. Even though many authors admit the importance of service-oriented computing (SOC) and see services as the base building block of future software architectures in general (for example [4] and [5]), the concept seems to be discussed mostly in the context of enterprise SOAs and often restricting it technically to the Web services standard family. In contrast to the state-of-the-art, the concept of service architecture presented in this paper is agnostic to any application domain or enabling technology.

2 Service Architecture Model

This section describes briefly the core elements of complete service architecture and the criteria for evaluating service platforms against the model. As depicted

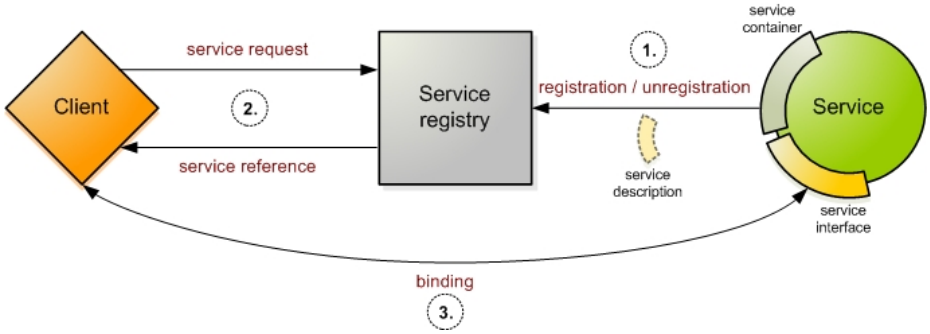


Fig. 1. Core elements of service architecture.

in Fig. 1, the elements are a service, its description, a service registry and a client.

A service implements some functionality which is used by clients through a public interface, described in the service description. The service is registered to the service registry by the container software that hosts the service implementation. During registration the service description is saved to the registry. A client can query the registry for discovering desired services. The registry responds by passing the client references to matching services. After that the client can contact a service directly and start using its functionality. The reference is an intermediary object that prevents creating unnecessary dependencies on the service before actually using it.

An important aspect of service architectures is that the entities are independent of each others, loosely coupled by help of explicit service descriptions. This establishes the core advantage of service architectures, making software very flexible and adaptive.

We have evaluated few existing service platforms in order to untangle how they conform to the model of service architecture. The evaluation criteria in Table 1 were drawn from the presented reference model. The evaluation results follow in the next section.

Many times service platforms support distribution, the Web services as an example. However, as service architectures are not limited to certain contexts or technologies, distribution is not a requirement at the level of service architecture. Need for distribution rather depends on the system under development and the service platform may provide it.

3 Service Platform Evaluation

3.1 OSGi Service Platform

The OSGi Service Platform (OSGi) is an industry-led specification for a dynamic component and service platform for Java. The platform overrides the

Table 1. Criteria for service architecture

Criterion	Description
Service registry	There is a centralized service registry that stores and delivers service descriptions, accessible by all entities.
Service description	Each service has a description that declares the service interface, access points and protocols.
Service references	The service registry responds to client requests with one or more service descriptions that might implement the desired functionality.
Service binding	A client can establish binding to a service based solely on the information provided in the service description.
Loose coupling	Client and service are independent of each other - binding occurs in the run-time, there is no compile-time linking.

standard Java class loading policies, enabling coordinated package visibility and dynamic resolving of dependencies. It also provides life-cycle management of the components, called bundles, which are ordinary Java archive files (JAR) with OSGi-specific headers in their manifest.

The platform has a service registry where bundles can register their functionality as services and use the services of other bundles. The service description defines the implemented Java interface and an optional dictionary with key-value properties. Any activated bundle within the platform has access to the registry. The framework gives in return an array of service reference objects that point to matching services in the registry. A service reference only contains the service description; the service instance can be requested from the framework with the reference. This two-phase service acquisition enables polling service information without creating any dependency yet on it. Once the framework returns the service instance, binding is established and the client can start using the service. The OSGi specification provides multiple means to adapt to service availability issues due to dynamic binding: service listeners, service trackers and declarative services. Despite of using these adaptation technologies the client should ensure that the service really is up and running by taking care it does not call a methods of a null-pointing service instance.

The OSGi specification is currently Java-specific. Services on other platforms need adapters to work with OSGi. The specification defines a UPnP base driver for enabling OSGi bundles to use and create UPnP services. There are also implementations for other bindings and Java Native Interface (JNI) technology is also available. Existing bindings from Java to other technologies can usually be converted to OSGi bundles and that has already been done to many technologies, including the Web services.

A distributed registry extension is in the making at the OSGi Alliance's Enterprise Expert Group which has recently published a draft of the new specification.[6] The draft proposes a framework for service interactions between multiple distinct OSGi platforms as well as between OSGi and non-OSGi service platforms. Similar extensions have also been proposed in the MidGate platform.[7]

The evaluation of OSGi platform against the service architecture criteria is presented in Table 2.

Table 2. Evaluation of OSGi Service Platform

Criterion	Realization
Service registry	**Fulfilled** – the service registry is available for any bundle in the platform and stores and delivers service descriptions.
Service description	**Fulfilled** – the description consists of the fully qualified interface class name and an optional set of properties.
Service references	**Fulfilled** – The references represent sufficient information of the service and help avoid unnecessary bindings.
Service binding	**Fulfilled** – A client bundle can obtain an instance of the service implementation with the service reference.
Loose coupling	**Fulfilled** – A client bundle can use any service by knowing only its interface at deployment time.

3.2 Web Services

The concept of Web services has many dissenting definitions; there is no unanimous agreement on which technologies exactly constitute the Web services, except that XML is the common base for them. The fundamental Web services technologies could be categorized to service description, service discovery, and communication technologies. There exist multiple technologies for each of these categories with possibly differing emphases.

The Web Services Interoperability Organization (WS-I) has published a set of profiles that mandate using certain versions of certain technologies in certain ways as providing for guaranteed interoperability of similarly profiled Web services. WS-I's Basic Profiles promote using Universal Description, Discovery and Integration (UDDI) as the service registry, Web Services Description Language (WSDL) for service descriptions and SOAP[1] as the messaging protocol between all entities.[8]

There are also other suitable technologies used for Web services, especially for messaging and service registries but not that many for service description, WSDL being almost unanimously agreed on. Available registry technologies comprise for example the Registry/Repository part of the Electronic Business using eXtended Markup Language (ebXML R/R), the Dublin Core Metadata Registry (DCMI Registry) and the ISO/IEC 11179 Metadata Registry (MDR) standard. In messaging the XML-RPC protocol aims to provide a more lightweight and straightforward alternative to SOAP and has gained some adoption.[9]

[1] SOAP originally stood for Simple Object Access Protocol. Since version 1.2 the name is no more considered an abbreviation.

The Web services technologies have played an important part as technological enablers in the research and applications of service-oriented architectures. In our analysis it is seen as a standards family that comprises a rather loosely defined bunch of XML-based technologies that can be used to realize platform-independent service architectures in terms of service description, service discovery and communication. For simplicity, we focus in this analysis on the technologies promoted by the WS-I Basic Profiles.

An UDDI-based registry provides the service registration and inquiry functions through SOAP APIs with XML-based request bodies. Each service is granted a universally unique identifier (UUID), against which the clients can request its description. The UDDI specification defines query patterns for how to find the desired services.[10]

A WSDL description provides the client with complete information of a service in three parts: the service interface, the supported communication protocols and their corresponding endpoints. The interface is described as an abstract collection of operations and the messages and data types used in them. The operations are mapped to one or more protocols, for example SOAP messages. Furthermore, a concrete endpoint is specified for each of these protocols.

SOAP is an application layer protocol, usually carried by another application layer protocol, the HTTP, for messaging between distributed parties. It can be utilized as RPC or document-oriented messaging. SOAP messages, called envelopes, consist of a body element and an optional header element and their content is defined by application, except of fault messages.

The presented Web services technologies provide a distributed service framework that provides full advantage of all the key entities of service architecture: service registry, services, service description and clients. There are no restrictions on implementation technologies as long as they support XML and networking. The rather loose framework widens the usage possibilities of Web services and allows for having fairly varying service implementations. WSDL seems to be the only widely fixed technology in the family. Because the binding protocols and registry standards are not similarly fixed to any single technologies, there could be interoperability issues; a Web service can be consumed only by those clients who support the announced binding protocols. However, SOAP over HTTP is a widespread form of communication between the Web services entities. The registry side seems to be dominated by UDDI but ebXML has also gained popularity. It has also been claimed that these two have differing intentions in the registry market.[11] Other authors suggests that despite the differing focuses the technical similarities still provide for interoperability among them [12] and they have developed a framework that unifies access to multiple registries.[13] The WS-I profiles guarantee interoperability of all Web services that conform to a certain profile. Table 3 summarizes the evaluation.

3.3 Multimedia Home Platform

Multimedia Home Platform (MHP) is a specification by the Digital Video Broadcasting Project (DVB) that defines middleware for digital television terminals

Table 3. Evaluation of Web services

Criterion	Realization
Service registry	**Fulfilled** – A UDDI (or other) service registry stores the WSDLs and is accessible with SOAP messages.
Service description	**Fulfilled** – Each service provides a WSDL description with interface, protocol and endpoint information.
Service references	**Fulfilled** – The WSDLs are also references to services as they include all the necessary service information.
Service binding	**Fulfilled** – Client can establish binding with a defined service endpoint using the defined protocol, like SOAP.
Loose coupling	**Fulfilled** – All the entities may reside on different hosts and heterogeneous environments.

in order to enable enhanced broadcasting and interactivity on digital television. The middleware is based on the Java Personal Basis Profile, a Java configuration for embedded devices.

The MHP specification defines the requirements for terminal devices in three profiles which makes it possible to implement the middleware in various hardware setups. The basic profile, Enhanced Broadcast, sets requirements for receiving multimedia, data streams and applications from the broadcast and running applications in the DTV terminal. The Interactive Broadcast profile adds a requirement for an IP-based interaction channel and its management. The Internet Access profile on top of the lower profiles provides a web browser, email and newsgroup client, and support for HTTP 1.1 protocol and IP multicasting.[14]

An MHP application, called Xlet, is provided as a part of a service that may include video, audio and data streams for a particular broadcasting channel. The application is run in a Java Virtual Machine (JVM) on the terminal device and is provided with APIs for accessing the resources of the device and the channel.

The Inter-Xlet Communication (IXC) introduced by the Java Personal Basis Profile forms the basis of the service platform in the MHP. The service registry is provided through a framework class which is available to all Xlets. The registry provides static methods for adding, updating and removing services as well as for looking up and listing them. No explicit service description is provided for the registry; services, which are normal Java objects, are registered with a freely defined binding name under the container Xlet's own namespace. Thus the service client needs to know in advance the service interface. As there is no means to sort out if multiple services implement the same interface, the client has to know in advance also the registration name of a certain service, which logically corresponds to compile-time linking, breaking the principle of loose coupling. Service references are not used but service request returns the service instance.

The MHP does not provide a platform for service architecture, partially due to the restrictions imposed by the specialized domain of the platform (digital television) but most fundamentally due to the lack of a proper service description and references and the consequent tight coupling between clients and services. Table 4 summarises the evaluation.

Table 4. Evaluation of Multimedia Home Platform

Criterion	Realization
Service registry	**Fulfilled** – The IXC registry provides functions for adding, updating and removing services and for browsing and requesting them.
Service description	**Not fulfilled** – Service description states only the service name without any reference to its interface.
Service references	**Not fulfilled** – Without references clients cannot examine service properties before binding.
Service binding	**Fulfilled** – Service binding is established right when the registry responds to service request successfully.
Loose coupling	**Not fulfilled** – Tight coupling due to poor service description.

4 Discussion

4.1 Service Architecture Model

The model of service architecture we present in this study is intentionally simplified; it represents the core elements but does not take into account all possible aspects or requirements of concrete service-based systems. For example when there are multiple implementations of a single service, it is not self-evident who authors the service description. The intention of this study, however, is to provide the basic building blocks that we think are obligatory in any feasible service architecture. Leaving out the service registry, for example, would require predeployment wiring of clients and services. Without service descriptions a tighter coupling between clients and services would be needed as sufficient information of service properties, interface and access information cannot be obtained at run-time. These lacks would lead to less flexible software.

Making clients depend on service interfaces rather than directly on the components behind them makes the system much more flexible and adaptive, as the clients do not have to declare any compile-time links to specific components. As long as a service's interface stays untouched, its implementation can be modified without implications on the client side. This is one important advantage of applying service architectures also within compact, possibly embedded systems; changes in service implementations do not necessarily incur the need to modify and rebuild the whole system and in many cases, the service providers can even update their services in the runtime because the clients are able to (re-)establish bindings in the runtime.

Another advantage of service architectures is that clients can possibly choose among multiple service providers if one provider suits better than the others. Or, if one service provider draws its service from the registry, a client can maybe switch to another provider and continue its operations.

This paper is based on an earlier study that concerns the same issues within the context of digital home.[15] In contrast to that work, this paper takes a more context-agnostic approach to service architectures. The earlier work presents also

a case study of interworking multiple service platforms, which unfortunately was not possible to fit into this paper. The earlier work is available in electronic form from the first author.

SOAs are a good and established approach in their own context of enterprise information systems and possibly other large scale, open software systems. The research of service-oriented computing could, however, be widened with a more extensive applying of service-orientation in various contexts, including small scale and closed systems.

4.2 Service Platforms

There are significant technical differences in the analyzed service platforms and they have differing intentions: OSGi provides a middleware layer on single-host systems that need a dynamic service environment; MHP is purposed as a platform for digital television broadcasters and users to interact; Web services technologies can be seen as an enabler for building distributed platform-independent, service-oriented systems and platforms.

Concerning OSGi, the current specification is intended for single-host implementations and does not support built-in distribution. The current specification defines a UPnP driver for opening access to UPnP services. Many other binding technologies like the Apache Axis Web services stack[2] have also been 'bundlified' to OSGi. However, these distribution models do not allow transparent distribution of OSGi services to multiple hosts but require introducing an intermediary component or service model like WSDL descriptions and SOAP endpoints. This is how distribution is implemented for example in the Newton framework that uses the Service Component Architecture (SCA) initiated by the OSOA Collaboration[3] as an intermediary component model in distribution. An intermediary component model will likely make client and service implementations more complex and have an effect on distribution transparency. Authors of [16] try to address this with their R-OSGi. Kang et al. have proposed an RMI-based extension for a distributed OSGi service registry.[17] There is also an ongoing work to include an implementation-neutral distribution framework in the next version of the formal specification, of which a draft has been recently published.[6]

The Web services interoperability issues caused for example by multiple registry standards could be solved by enhancing registry interoperability and unified access to differing registry standards, as proposed and realized in [12] and [13]. Also the profiles mandated by the WS-I are a good way to enhance interoperability.

Considering MHP, the IXC architecture could be enhanced by requiring more sufficient service description with service interface as this would lower the coupling between the service and the client. It might be fruitful to analyze how the OSGi service framework could be utilized as the basis for the MHP software stack. That would readily provide a good and fully functional, mature service

[2] http://ws.apache.org/axis2/
[3] http://www.osoa.org

platform that is already suitable for embedded systems. Java and JVM as the runtime platform are already common denominators for both, and the OSGi bundles and Xlets share a similar application model with a framework-controlled life cycle. The authors of [18] discuss the need for integrating these two platforms in order to provide a single residential gateway for connecting and managing home networks and devices. They propose a solution for integrating these technologies and introduce a new application model, called XbundLET, which combines the Xlet and OSGi bundle model and acts as a proxy between the two frameworks. The authors primarily see OSGi as a service-oriented framework for connecting home appliances and do not recognize it as "universal middleware" as declared by the OSGi Alliance itself after a shift from the original idea of an open services gateway framework. This universality is proven by numerous realized use cases from various fields, ranging from the automotive industry to mobile devices, enterprise systems and application servers. In this light we do not see great barriers in shifting MHP from the Java PBP to OSGi. Such a shift would also have the advantage of making the architecture extensible, which may lengthen the life span of the terminal software.

5 Conclusion

In this paper we have presented a simple model for service architectures based on the state-of-the-art research and technologies. The model has four fundamental entities: client, service, service description and service registry. Services are registered to the registry with a description that presents sufficient information of the service to the clients. Clients can find suitable services from the registry, examine their properties before calling them and make bindings to them based solely on the information in service descriptions.

The criteria for service architectures were drawn from the model and three existing service platforms analyzed against these criteria. The OSGi Service Platform and the Web services family fulfill the presented criteria and thus can be considered as platforms for service architectures. MHP did not fulfill the requirements as it lacked a sufficient service description which is crucial for achieving loose coupling of entities within a dynamic environment.

This study provides an introductory insight to the vast research area and usage possibilities of service architectures in various contexts. There are fundamental differences in service-oriented architectures (SOA) and service architectures, mostly in their scope, as the latter provides a more abstract model of applying service-orientation. SOA-related research and solutions are often aimed at large software systems, although, as shown by this study, service-orientation as such can be applied in other contexts too, whether small or large in their scale. The current state-of-the-art is pretty scattered and despite SOA being such a popular field today, a clear field of research on service architectures in particular has not yet emerged. There is a wide door open for service architectures as a new theme in service-oriented computing.

References

1. Erl T.: Service-Oriented Architecture: Concepts, Technology, and Design. Ch. 8: Principles of Service-Orientation. Prentice Hall, (2005)
2. Tsai W. T.: Service-oriented system engineering: a new paradigm. (2005)
3. Open Group, Definition of SOA, http://opengroup.org/projects/soa/doc.tpl?gdid=10632. (checked: 16.9.2008)
4. Bennett, K., Layzell, P., Budgen, D., Brereton, P., Macaulay, L., Munro, M.: Service-Based Software: The Future for Flexible Software. In: APSEC 2000: Proceedings of the Seventh Asia-Pacific Software Engineering Conference. Singapore. pp. 214-221. (2000)
5. Papazoglou, M. P., Traverso, P., Dustdar, S., Leymann, F., Krämer, B. J.: Service-Oriented Computing: A Research Roadmap. In: Dagstuhl Seminar Proceedings on Service Oriented Computing (SOC). (2006)
6. OSGi Alliance: OSGi Service Platform Release 4 - Version 4.2 - Early Draft. (2008)
7. Pakkala D.: MidGate: middleware platform for service gateway based distributed systems. VTT Publications, http://www.vtt.fi/inf/pdf/publications/2004/P519.pdf (2005)
8. WS-I, Deliverables from the Basic Profile Working Group, http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile. (checked: 16.9.2008)
9. XML-RPC Home page, http://www.xmlrpc.com. (checked: 16.9.2008)
10. UDDI Version 2.04 API Specification, http://uddi.org/pubs/ProgrammersAPI_v2.htm. (checked: 16.9.2008)
11. Understanding ebXML, UDDI, XML/EDI. XML.org, http://www.xml.org/xml/feature_articles/2000_1107_miller.shtml (checked: 16.9.2008)
12. Al-Masri, E., Mahmoud, Q. H.: Interoperability among Service Registry Standards. IEEE Internet Comput., Vol. 11, Issue 3, 74-77. (2007)
13. Al-Masri, E., Mahmoud, Q. H.: A Framework for Efficient Discovery of Web Services Across Heterogeneous Registries. In: CCNC 2007. 4th IEEE Consumer Communications and Networking Conference. Las Vegas, Nevada, USA. pp. 415-419. (2007)
14. DVB Project: Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.2. http://www.mhp.org/specs/a107_mhp_12.zip (2007)
15. Mäki, M.: Feasibility of Service Architectures for Digital Home Systems. Master's Thesis. University of Oulu, Finland. (2008)
16. Rellermeyer, J. S., Alonso, G., Roscoe, T.: R-OSGi: Distributed Applications through Software Modularization. In: Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference. California, US. (2007)
17. Kang, K., Lee, J., Choi, H.: Extended Service Registry for Distributed Computing Support in OSGi Architecture. In: 8th International Conference Advanced Communication Technology, 2006. ICACT 2006. pp. 1631-1634. (2006)
18. Vilas, A., Redondo, R., Cabrer, M., Arias, J., Solla, A., Duque, J., Nores, M., Fernández, Y.: MHP-OSGi Convergence: A New Model for Open Residential Gateways. Softw. Pract. Exper., Vol. 36, Issue 13, 1421-1442. (2006)

Author Index